Principles of Successful Testing and Testing Techniques

Instructor: C. Pu (Ph.D., Assistant Professor)

Lecture 01

puc@marshall.edu

The Economics of Insecure Software

- Security measurements are about both the *specific technical issues* and how these *issues affect the economics of software*
  - Most technical people will at least
    - understand the basic issues
    - or have a deeper understanding of the vulnerabilities
  - Few are able to translate that technical knowledge into monetary terms and quantify the *potential cost* of vulnerabilities to the application owner's business
  - Until this happens, CIOs will not be able to develop an accurate return on security investment and, subsequently, assign appropriate budgets for software security

The Economics of Insecure Software

- A survey on the cost of insecure software to the US economy due to *inadequate software testing* (NIST, June 2002)
  - a better testing infrastructure would save more than a third of these costs, or about $22 billion a year

- The relationships between economics and security have been studied by academic researchers
  - https://www.cl.cam.ac.uk/~rja14/econsec.html

The Economics of Insecure Software

- People are encouraged to measure security *throughout the entire development process*
  - relate the cost of insecure software to the impact it has on the business
  - consequently develop appropriate business processes and assign resources to manage the risk

- Remember that measuring and testing web applications is even *more critical* than for other software
  - web applications are exposed to millions of users through the Internet

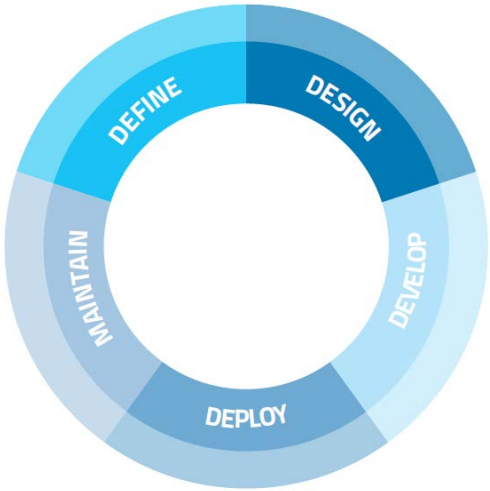
What is Testing?

- During the development life cycle of a web application, many things need to be tested, but what does testing actually mean?
- The Dictionary describes testing as:
  - to put to test or proof
  - to undergo a test
  - to be assigned a standing or evaluation based on tests
- In software development, testing is a process of comparing the state of a system or application against a set of criteria

When to Test?

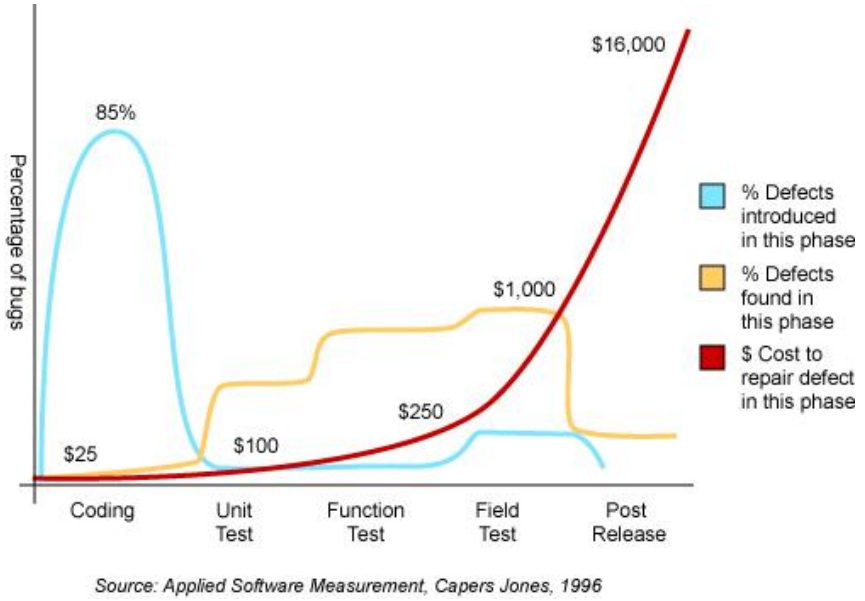
- Most people today don't test software until it has already been created and is in the *deployment phase* of its life cycle
  - very ineffective and cost-prohibitive practice

- One of the best methods to prevent security bugs from appearing in production applications
  - improve the *Software Development Life Cycle (SDLC)* by *including security in each of its phases*



When to Test?

- A generic SDLC model as well as the (estimated) increasing cost of fixing security bugs in such a model



Source: Applied Software Measurement, Capers Jones, 1996

- Companies should inspect their overall SDLC to ensure that security is an integral part of the development process
- SDLCs should include security tests to ensure security is adequately covered and controls are effective throughout the development process

What to Test?

- It can be helpful to think of software development as a combination of *people*, *process*, and *technology*

- An effective testing program should have components that test:
  - *People*
    - to ensure that there is adequate education and awareness
  - *Process*
    - to ensure that there are adequate policies and standards and that people know how to follow these policies
  - *Technology*
    - to ensure that the process has been effective in its implementation

Principles of Testing: There is No Silver Bullet

- A security scanner or application firewall will provide many defenses against attack or identify a multitude of problems?
  - there is no silver bullet to the problem of insecure software
  - application security assessment software is generally immature and ineffective at in-depth assessments or providing adequate test coverage
- Remember that security is a process and not a product

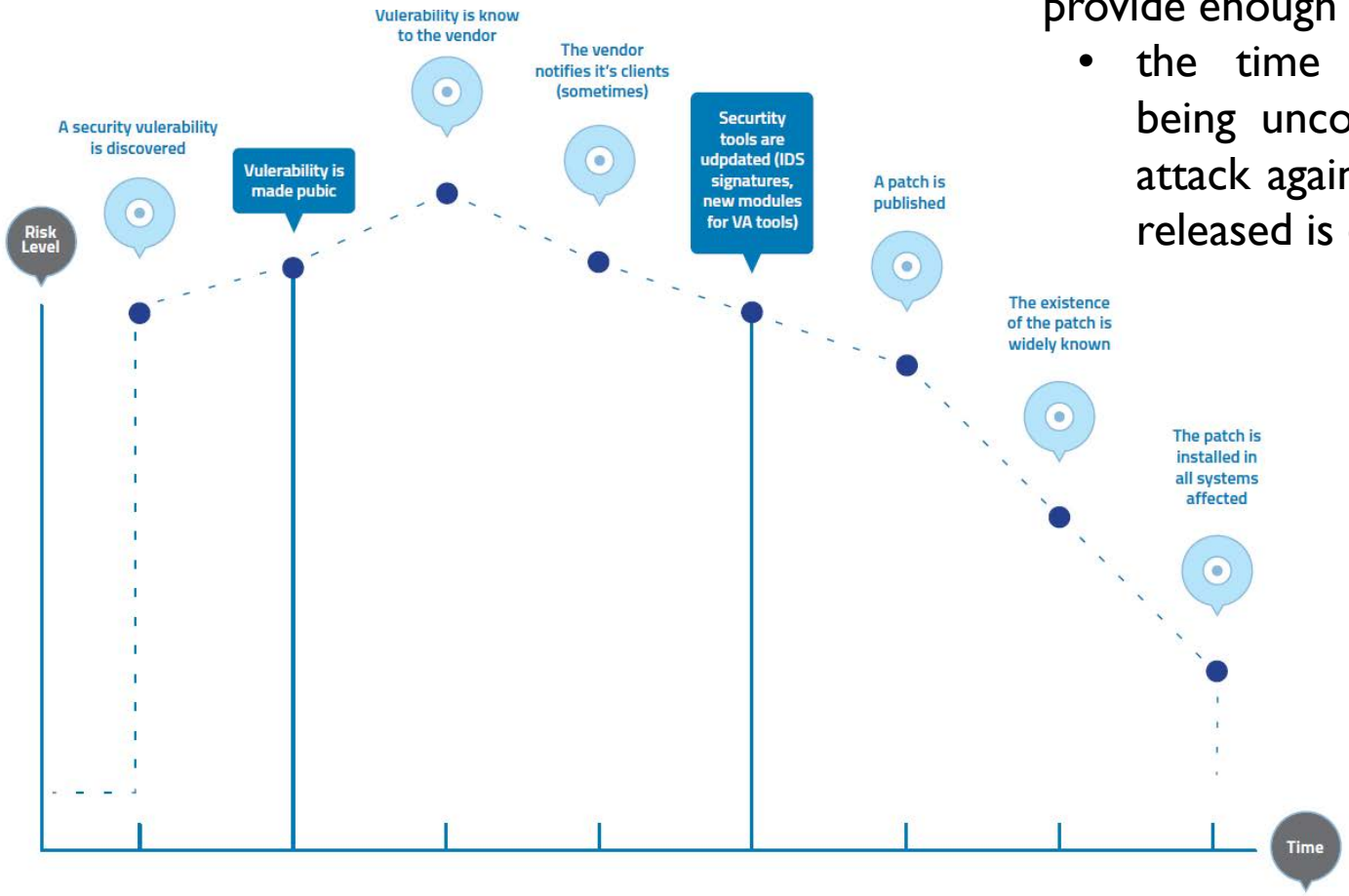
Principles of Testing: Think Strategically, Not Tactically

- The fallacy of the patch-and-penetrate model
  - pervasive in information security during the 1990's

- The patch-and-penetrate model involves fixing a reported bug, but without proper investigation of the root cause
  - The evolution of vulnerabilities in common software used worldwide has shown the ineffectiveness of this model

Principles of Testing: Think Strategically, Not Tactically

Window of Vulnerability



- With the reaction time of attacker, the typical window of vulnerability does not provide enough time for patch installation
  - the time between a vulnerability being uncovered and an automated attack against it being developed and released is decreasing every year

Principles of Testing: Think Strategically, Not Tactically

- Several *incorrect* assumptions in the patch-and-penetrate model
  - patches interfere with normal operations and might break existing applications
  - all users are aware of newly released patches

- Consequently,
  - not all users of a product will apply patches
    - either because they think patching may interfere with how the software works
    - or because they lack knowledge about the existence of the patch

Principles of Testing: Think Strategically, Not Tactically

- It is essential to build security into the Software Development Life Cycle (SDLC) to prevent reoccurring security problems within an application
- Developers can build security into the SDLC by developing standards, policies, and guidelines that fit and work within the development methodology
- Threat modeling and other techniques should be used to help assign appropriate resources to those parts of a system that are most at risk

Principles of Testing: Test Early and Test Often

- When a bug is detected early within the SDLC it can be addressed faster and at a lower cost
  - A security bug is no different from a functional or performance-based bug in this regard
  - A key step in making this possible is to educate the development and QA teams about common security issues and the ways to detect and prevent them
  - Education in security testing also helps developers acquire the appropriate mindset to test an application from an attacker's perspective
    - This allows each organization to consider security issues as part of their existing responsibilities

Principles of Testing: Understand the Scope of Security

- It is important to know how much security a given project will require
- The information and assets that are to be protected should be given a classification that states how they are to be handled (e.g., confidential, secret, top secret)
- Discussions should occur with legal council to ensure that any specific security requirements will be met
  - In the USA, requirements might come from federal regulations, such as the Gramm-Leach-Bliley Act, or from state laws, such as the California SB-1386

Principles of Testing: Develop the Right Mindset

- Successfully testing an application for security vulnerabilities requires *thinking "outside of the box."*
- Normal use cases will test the normal behavior of the application when a user is using it in the manner that is expected
- Good security testing requires going beyond what is expected and thinking like an attacker who is trying to break the application
  - Creative thinking can help to determine what unexpected data may cause an application to fail in an insecure manner
  - It can also help find what assumptions made by web developers are not always true and how they can be subverted

Principles of Testing: Understand the Subject

- One of the first major initiatives in any good security program should be to require accurate documentation of the application
  - architecture
  - data-flow diagrams
  - use cases
  - etc.
- The technical specification and application documents should include information that lists not only the desired use cases, but also any specifically disallowed use case
- Finally, it is good to have at least a basic security infrastructure that allows the monitoring and trending of attacks against an organization's applications and network

Principles of Testing: Use the Right Tools

- While we have already stated that there is no silver bullet tool, tools do play a critical role in the overall security program
- There is a range of open source and commercial tools that can automate many routine security tasks
- These tools can simplify and speed up the security process by assisting security personnel in their tasks
- However, it is important to understand exactly what these tools can and cannot do so that they are not oversold or used incorrectly

Principles of Testing: Use Source Code When Available

- While *black box penetration test* results can be impressive and useful to demonstrate how vulnerabilities are exposed in a production environment, they are not the most effective or efficient way to secure an application
- It is difficult for dynamic testing to test the entire code base, particularly if many nested conditional statements exist
- If the source code for the application is available, it should be given to the security staff to assist them while performing their review.
- It is possible to discover vulnerabilities within the application source that would be missed during a black box engagement

Principles of Testing: Document the Test Results

- To conclude the testing process, it is important to produce a formal record of what testing actions were taken, by whom, when they were performed, and details of the test findings
  - It is wise to agree on an acceptable format for the report which is useful to all concerned parties,
    - developers
    - project management
    - business owners
    - IT department
    - audit

Principles of Testing: Document the Test Results

- The report should be clear to the business owner in identifying where risks exist and sufficient to get their backing for subsequent mitigation actions
- The report should also be clear to the developer in pin-pointing the exact function that is affected by the vulnerability and associated recommendations for resolving issues in a language that the developer will understand
- The report should also allow another security tester to reproduce the results
- Writing the report should not be overly burdensome on the security tester themselves
- Using a security test report template can save time and ensure that results are documented accurately and consistently, and are in a format that is suitable for the audience

Testing Techniques

- Testing techniques
  - Manual Inspections & Reviews
  - Threat Modeling
  - Code Review
  - Penetration Testing

Testing Techniques: Manual Inspections & Reviews

- Manual inspections are *human reviews* that typically test the security implications of people, policies, and processes
  - Manual inspections can also include inspection of technology decisions such as architectural designs
  - They are usually conducted by analyzing documentation or performing interviews with the designers or system owners

Testing Techniques: Manual Inspections & Reviews

- While the concept of manual inspections and human reviews is simple, they can be among the most powerful and effective techniques available
- By asking someone how something works and why it was implemented in a specific way, the tester can quickly determine if any security concerns are likely to be evident
- Manual inspections and reviews are one of the few ways to test the software development life-cycle process itself and to ensure that there is an adequate policy or skill set in place

Testing Techniques: Manual Inspections & Reviews

- As with many things in life, when conducting manual inspections and reviews it is recommended that a *trust-but-verify model* is adopted.
  - Not everything that the tester is shown or told will be accurate
- Manual reviews are particularly good for testing whether people understand the security process, have been made aware of policy, and have the appropriate skills to design or implement a secure application
- Other activities, including manually reviewing the documentation, secure coding policies, security requirements, and architectural designs, should all be accomplished using manual inspections

Testing Techniques: Manual Inspections & Reviews

- Advantages:
  - Requires no supporting technology
  - Can be applied to a variety of situations
  - Flexible
  - Promotes teamwork
  - Early in the SDLC
- Disadvantages:
  - Can be time consuming
  - Supporting material not always available
  - Requires significant human thought and skill to be effective

Testing Techniques: Threat Modeling

- Threat modeling has become a popular technique to help system designers think about the security threats that their systems and applications might face
  - threat modeling can be seen as risk assessment for applications
  - it enables the designer to develop mitigation strategies for potential vulnerabilities and helps them focus their inevitably limited resources and attention on the parts of the system that most require it
  - it is recommended that all applications have a threat model developed and documented
- Threat models should be created as early as possible in the SDLC, and should be revisited as the application evolves and development progresses

Testing Techniques: Threat Modeling

- To develop a threat model, it is recommend taking a simple approach that follows the NIST 800-30 standard for risk assessment.
- This approach involves:
  - Decomposing the application – use a process of manual inspection to understand how the application works, its assets, functionality, and connectivity.
  - Defining and classifying the assets – classify the assets into tangible and intangible assets and rank them according to business importance.
  - Exploring potential vulnerabilities - whether technical, operational, or management.
  - Exploring potential threats – develop a realistic view of potential attack vectors from an attacker's perspective, by using threat scenarios or attack trees.
  - Creating mitigation strategies – develop mitigating controls for each of the threats deemed to be realistic.

Testing Techniques: Threat Modeling

- The output from a threat model itself can vary but is typically a collection of lists and diagrams
- The OWASP Code Review Guide outlines an Application Threat Modeling methodology that can be used as a reference for the testing applications for potential security flaws in the design of the application
- There is no right or wrong way to develop threat models and perform information risk assessments on applications

Testing Techniques: Threat Modeling

- Advantages:
  - Practical attacker's view of the system
  - Flexible
  - Early in the SDLC
- Disadvantages:
  - Relatively new technique
  - Good threat models don't automatically mean good software

Testing Techniques: Source Code Review

- Source code review is the process of manually checking the source code of a web application for security issues.
  - As the popular saying goes "if you want to know what's really going on, go straight to the source."
- Many serious security vulnerabilities cannot be detected with any other form of analysis or testing
- Almost all security experts agree that there is no substitute for actually looking at the code
  - All the information for identifying security problems is there in the code somewhere
- Unlike testing third party closed software such as operating systems, when testing web applications (especially if they have been developed in-house) the source code should be made available for testing purposes

Testing Techniques: Source Code Review

- Many unintentional but significant security problems are also extremely difficult to discover with other forms of analysis or testing, such as penetration testing, making source code analysis the technique of choice for technical testing

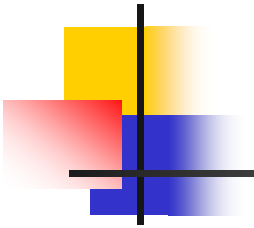
- With the source code, a tester can accurately determine what is happening (or is supposed to be happening) and remove the guess work of black box testing

Testing Techniques: Source Code Review

- Advantages:
  - Completeness and effectiveness
  - Accuracy
  - Fast (for competent reviewers)

- Disadvantages:
  - Requires highly skilled security developers
  - Cannot detect run-time errors easily
  - The source code actually deployed might differ from the one being analyzed

Testing Techniques: Penetration Testing



- Penetration testing has been a common technique used to test network security for many years
- It is also commonly known as *black box testing* or *ethical hacking*
- Penetration testing is essentially the "art" of testing a running application remotely to find security vulnerabilities, without knowing the inner workings of the application itself
- Typically, the penetration test team would have access to an application as if they were users
- The tester *acts like an attacker* and attempts to find and exploit vulnerabilities
- In many cases the tester will be given a valid account on the system

Testing Techniques: Penetration Testing

- Many people today use web application penetration testing as their primary security testing technique
- Whilst it certainly has its place in a testing program, we do not believe it should be considered as the primary or only testing technique

> "If you fail a penetration test you know you have a very bad problem indeed. If you pass a penetration test you do not know that you don't have a very bad problem". (Gary McGraw)

- Focused penetration testing can be useful in detecting if some specific vulnerabilities are actually fixed in the source code deployed on the web site

Testing Techniques: Penetration Testing

- Advantages:
  - Can be fast (and therefore cheap)
  - Requires a relatively lower skill-set than source code review
  - Tests the code that is actually being exposed

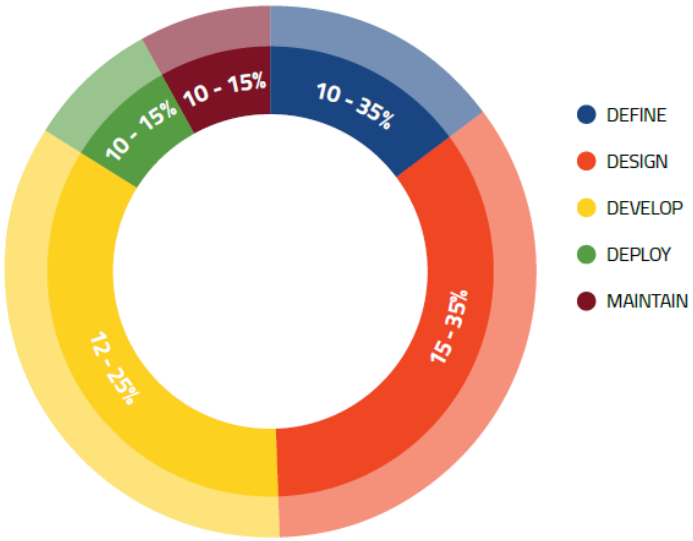
- Disadvantages:
  - Too late in the SDLC
  - Front impact testing only

Testing Techniques: The Need for a Balanced Approach

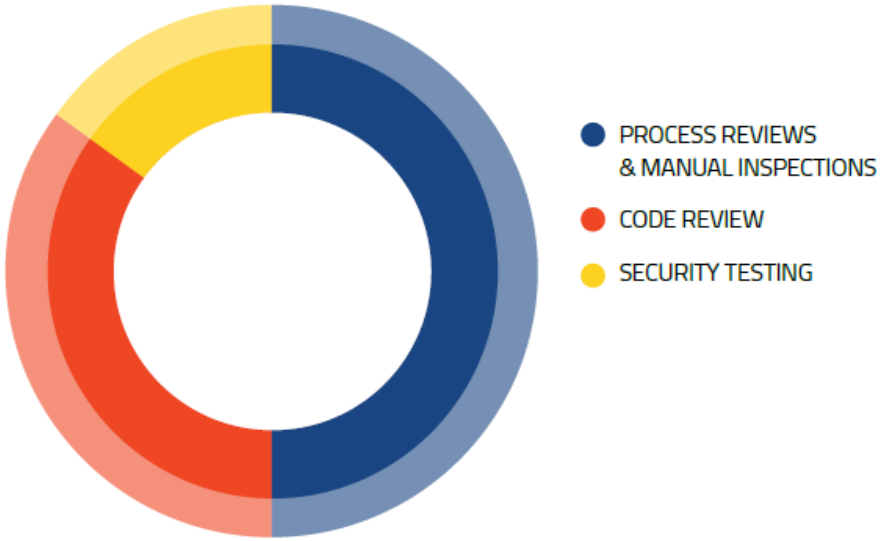
- With so many techniques and approaches to testing the security of web applications, it can be difficult to understand which techniques to use and when to use them
- The correct approach is a balanced approach that includes several techniques, from manual reviews to technical testing
  - A balanced approach should cover testing in all phases of the SDLC
  - This approach leverages the most appropriate techniques available depending on the current SDLC phase
- A balanced approach varies depending on many factors, such as the maturity of the testing process and corporate culture

Testing Techniques: The Need for a Balanced Approach

- It is recommended that a balanced testing framework should look something like the representations



Proportion of Test Effort in SDLC



Proportion of Test Effort According to Test Technique

Deriving Security Test Requirements

- To have a successful testing program, one must know what the testing objectives are
- These objectives are specified by the security requirements

Deriving Security Test Requirements: Testing Objectives

- One of the objectives of security testing is to validate that *security controls operate as expected*
  - This is documented via security requirements that describe the functionality of the security control
  - At a high level, this means proving confidentiality, integrity, and availability of the data as well as the service
- The other objective is to validate that *security controls are implemented with few or no vulnerabilities*
  - These are common vulnerabilities, such as the OWASP Top Ten, as well as vulnerabilities that have been previously identified with security assessments during the SDLC, such as threat modelling, source code analysis, and penetration test

Deriving Security Test Requirements: Security Requirements Documentation

- The first step in the documentation of security requirements is to understand the business requirements
  - A business requirement document can provide initial high-level information on the expected functionality of the application
  - For example,
    - the main purpose of an application may be to provide financial services to customers or to allow goods to be purchased from an on-line catalog
  - A security section of the business requirements should highlight the need to protect the customer data as well as to comply with applicable security documentation such as regulations, standards, and policies

Deriving Security Test Requirements: Security Requirements Documentation

- A general checklist of the applicable regulations, standards, and policies is a good preliminary security compliance analysis for web applications
  - For example,
    - compliance regulations can be identified by checking information about the business sector and the country or state where the application will operate
- Some of these compliance guidelines and regulations might translate into specific technical requirements for security controls
  - For example,
    - in the case of financial applications, the compliance with FFIEC guidelines for authentication requires that financial institutions implement applications that mitigate weak authentication risks with multi-layered security control and multi-factor authentication

Deriving Security Test Requirements: Security Requirements Documentation

- Applicable industry standards for security need also to be captured by the general security requirement checklist
  - For example,
    - in the case of applications that handle customer credit card data, the compliance with the PCI DSS standard forbids the storage of PINs and CVV2 data and requires that the merchant protect magnetic strip data in storage and transmission with encryption and on display by masking.
    - Such PCI DSS security requirements could be validated via source code analysis

Deriving Security Test Requirements: Security Requirements Documentation

- Another section of the checklist needs to enforce general requirements for compliance with the organization's information security standards and policies

- From the functional requirements perspective, requirements for the security control need to map to a specific section of the information security standards
  - For example,
    - a password complexity of six alphanumeric characters must be enforced by the authentication controls used by the application

Deriving Security Test Requirements: Threats and Countermeasures Taxonomies

- A threat and countermeasure classification, which takes into consideration root causes of vulnerabilities, is the critical factor in verifying that security controls are designed, coded, and built to mitigate the impact of the exposure of such vulnerabilities.
- In the case of web applications, the exposure of security controls to common vulnerabilities, such as the OWASP Top Ten, can be a good starting point to derive general security requirements.
- More specifically, the web application security frame provides a classification (e.g. taxonomy) of vulnerabilities that can be documented in different guidelines and standards and validated with security tests.
  - https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649461(v=pandp.10)?redirectedfrom=MSDN

Deriving Security Test Requirements: Threats and Countermeasures Taxonomies

- The focus of a threat and countermeasure categorization is to define security requirements in terms of the threats and the root cause of the vulnerability.
- A threat can be categorized by using STRIDE as Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.
- The root cause can be categorized as security flaw in design, a security bug in coding, or an issue due to insecure configuration.
- For example,
  - the root cause of weak authentication vulnerability might be the lack of mutual authentication when data crosses a trust boundary between the client and server tiers of the application.

Deriving Security Test Requirements: Security Testing and Risk Analysis

- Security requirements need to take into consideration the severity of the vulnerabilities to support a risk mitigation strategy

- Assuming that the organization maintains a repository of vulnerabilities found in applications (i.e, a vulnerability knowledge base), the security issues can be reported by type, issue, mitigation, root cause, and mapped to the applications where they are found

- Such a vulnerability knowledge base can also be used to establish a metrics to analyze the effectiveness of the security tests throughout the SDLC

Deriving Security Test Requirements: Security Testing and Risk Analysis

- For example, consider an input validation issue, such as a SQL injection, which was identified via source code analysis and reported with a coding error root cause and input validation vulnerability type.
- The exposure of such vulnerability can be assessed via a penetration test, by probing input fields with several SQL injection attack vectors.
- This test might validate that special characters are filtered before hitting the database and mitigate the vulnerability.
- By combining the results of source code analysis and penetration testing it is possible to determine the likelihood and exposure of the vulnerability and calculate the risk rating of the vulnerability.
- By reporting vulnerability risk ratings in the findings (e.g., test report) it is possible to decide on the mitigation strategy.
- For example, high and medium risk vulnerabilities can be prioritized for remediation, while low risk can be fixed in further releases.