بسم الله الرحمن الرحيم

KHAWAJAGAN Technologies

Secure Software Design and Development

SP 2025

CYC386

KHAWAJAGAN Technologies

WEEK 005 – 14-MAR-2025

FAISAL SHAHZAD

SP 2025

Security Test Cases

Test Case

- A security test case is a document that describes an input, action, or event that is expected to produce a predictable response.
- The fundamental aim of all test cases is to find out if a specified feature in a system or software product is working properly.
- A test case should contain particulars such as test case identifier, test case name, objective, test conditions/ setup, input data requirements, steps, and expected results.
- The process of developing test cases can help find problems in the requirements or design of an application, since it requires completely thinking through the operation of the application.
- For this reason, it's a useful habit to prepare test cases as early in the development process as possible.

Test Case

- Testing involves examining a system or application under controlled conditions and then evaluating the results.
  - For example, the tester asks, "If the user is in interface A of the application while using hardware B and does C, then D should happen."
- From a practical standpoint, the controlled conditions ought to include both normal and abnormal conditions.
- In that respect, testing should intentionally try to make things go wrong in the product to determine what undesirable events might occur when they shouldn't, or which desirable events don't happen when they should.

Test Case

- There are numerous forms of testing, all with a slightly different purpose and focus, however, generally, these fall into two categories:
  - White box testing
  - Black box testing
- Testing that tries to exercise as much of the code as possible within some set of resource constraints is called white-box testing.
- Techniques that do not consider the code's structure when test cases are selected are called black-box techniques.
- What we are going to do is examine the most common approaches within those categories, keeping in mind that even the explicit methodologies we are going to discuss are implemented in different ways depending on the integrity level required for the software product being tested.

SP 2025

Attack Surface Evaluation

Attack Surface

- The attack surface for a system is technically the set of points on the boundary of a system, a system element, or an environment where an attacker can try to enter and cause an effect on, or extract data from, that system, system element, or environment.
- In most software systems, this means the place where the data or information enters the system, either legitimately or in an unauthorized fashion.
- During the design phase, an estimate of the risks and the mitigation efforts associated with the risks is performed.
- The various entry points and the risks associated with them should be examined. Based on the results of this design, the system is developed, and during development, the actual system design goals may or may not have been met.
- Testing the code for obvious failures at each step along the way provides significant information as to which design elements were not met.

Attack Surface

- It is important to document the actual attack surface throughout the development process.
- Testing the elements and updating the attack surface model provide the development team with feedback, ensuring that the design attack surface objectives are being met through the development process.
- Testing elements such as the level of code accessible by untrusted users, the quantity of elevated privilege code, and the implementation of mitigation plans detailed in the threat model is essential in ensuring that the security objectives are being met through the development process.

Attack Surface Evaluation

- Microsoft has developed and released a tool called the attack surface analyzer, which is designed to measure the security impact of an application on a Windows environment.
- Acting as a sophisticated scanner, the tool can detect the changes that occur to the underlying Windows OS when an application is installed.
- Designed to specifically look for and alert on issues that have been shown to cause security weaknesses, the attack surface analyzer enables a development team or an end user to do the following:
  - View changes in the Windows attack surface resulting from the installation of the application
  - Assess the aggregate attack surface change associated with the application in the enterprise environment
  - Evaluate the risk to the platform where the application is proposed to exist
  - Provide incident response teams detailed information associated with a Windows platform

Attack Surface Evaluation

- One of the advantages of the attack surface analyzer is that it operates independently of the application that is under test.
- The attack surface analyzer scans the Windows OS environment and provides actionable information on the security implications of an application when installed on a Windows platform.
- For this reason, it is an ideal scanner for final security testing as part of the secure development lifecycle (SDL) for applications targeted to Windows environments.

SP 2025

Penetration Testing

Penetration Testing

- Penetration testing, sometimes called pen testing, is an active form of examining the system for weaknesses and vulnerabilities.
- While scanning activities are passive in nature, penetration testing is more active.
- Vulnerability scanners operate in a sweep, looking for vulnerabilities using limited intelligence; penetration testing harnesses the power of human intellect to make a more targeted examination.
- Penetration testers attack a system using information gathered from it and expert knowledge in how weaknesses can exist in systems.

Penetration Testing

- Penetration testing is designed to mimic the attacker's ethos and methodology, with the objective of finding issues before an adversary does.
- It is a highly structured and systematic method of exploring a system and finding and attacking weaknesses.
- Penetration testing is a valuable part of the SDL process. It can dissect a program and determine if the planned mitigations are effective.
- Pen testing can discover vulnerabilities that were not thought of or mitigated by the development team.
- It can be done with a white-, black-, or gray-box testing mode.

Penetration Testing

- The penetration testing process begins with specific objectives being set out for the tester to explore.
- For software under development, these could be input validation vulnerabilities, configuration vulnerabilities, and vulnerabilities introduced to the host platform during deployment.
- Based on the objectives, a test plan is created and executed to verify that the software is free of known vulnerabilities.
- As the testers probe the software, they take notes of the errors and responses, using this information to shape subsequent tests.

Penetration Testing

- Penetration testing is a slow and methodical process, with each step and the results being validated.
- The records of the tests should demonstrate a reproducible situation where the potential vulnerabilities are disclosed.
- This information can give the development team a clear picture of what was found so that the true root causes can be identified and fixed.

SP 2025

Common Penetration Testing Methods

Common PenTest Methods

- Several common methods are employed when testing to make different determinations concerning software fitness.
  - **Fuzzing** is a technique used to find specific errors associated with input validation.
  - **Scanning** is another method used to look for specific conditions that result in undesired states or outcomes.
  - **Simulations** are a means of testing how the software performs under typical operating conditions.
  - **Failure mode testing** checks that known failure modes are understood and caught.
  - **Cryptographic testing** is important because of the highly technical nature of cryptography.
  - **Regression testing** is used to ensure that errors that are found are tested against different versions of the code base that are in production.
  - **Integration testing** is done to manage interfaces with external data sources, and continuous testing is performed.

SP 2025

Fuzzing

Fuzzing

- Fuzz testing is a brute-force method of addressing input validation issues and vulnerabilities.
- Fuzzing a program includes applying large numbers of inputs to determine which ones cause faults and which ones might be vulnerable to exploitation.
- Fuzz testing can be applied to anywhere data is exchanged to verify that input validation is being performed properly.
- Network protocols can be fuzzed, file protocols can be fuzzed, and web protocols can be fuzzed.
- The vast majority of browser errors are found via fuzzing.
- Fuzz testing works well in white-, black-, or gray-box testing, as it can be independent of the specifics of the application under test.

Fuzzing

- Fuzz testing works by sending a multitude of input signals and seeing how the program handles them. Specifically, malformed inputs can be used to vary parser operation and to check for memory leaks, buffer overflows, and a wide range of input validation issues.
- Since input validation errors are one of the top issues in software vulnerabilities, fuzzing is the best method of testing against these issues, such as cross-site scripting and injection vulnerabilities.
- There are several ways to classify fuzz testing.
  - Smart testing uses knowledge of what could go wrong and creates malformed inputs with this knowledge.
  - Dumb testing just uses random inputs.
  - Generation-based fuzz testing uses the specifications of input streams to determine the data streams that are to be used in testing.
  - Mutation-based fuzzers take known good traffic and mutate it in specific ways to create new input streams for testing.
- Each of these has its advantages, and the typical fuzzing environment involves both used together.

Fuzzing

- Fuzz testing works by sending a multitude of input signals and seeing how the program handles them. Specifically, malformed inputs can be used to vary parser operation and to check for memory leaks, buffer overflows, and a wide range of input validation issues.
- Since input validation errors are one of the top issues in software vulnerabilities, fuzzing is the best method of testing against these issues, such as cross-site scripting and injection vulnerabilities.
- There are several ways to classify fuzz testing.
  - Smart testing uses knowledge of what could go wrong and creates malformed inputs with this knowledge.
  - Dumb testing just uses random inputs.
  - Generation-based fuzz testing uses the specifications of input streams to determine the data streams that are to be used in testing.
  - Mutation-based fuzzers take known good traffic and mutate it in specific ways to create new input streams for testing.
- Each of these has its advantages, and the typical fuzzing environment involves both used together.

SP 2025

Scanning

Scanning

- Scanning is automated enumeration of specific characteristics of an application or network.
- These characteristics can be of many different forms, from operating characteristics to weaknesses or vulnerabilities and are as follows:
  - Network scans can be performed to identify network devices available and responsive on the network.
  - Systems can be scanned to determine the specific operating system (OS) in place, a process known as OS fingerprinting.
  - Vulnerability scanners can scan applications to determine if specific vulnerabilities are present.
- Scanning can be used in software development to characterize an application on a target platform.
- It can provide the development team with a wealth of information as to how a system will behave when deployed into production.
- There are numerous security standards, including the Payment Card Industry Data Security Standard (PCI DSS), that have provisions requiring the use of scanners to identify weaknesses and vulnerabilities in enterprise platforms.

Scanning

- The development team should take note that enterprises will be scanning the application as installed in the enterprise.
- Gaining an understanding of the footprint and security implications of an application before shipping will help the team to identify potential issues before they are discovered by customers.
- Scanners have been developed to search for a variety of specific conditions.
- There are scanners that can search code bases for patterns that are indicative of elements of the OWASP Top 10 and the SANS Top 25 lists.
- There are scanners tuned to produce reports for PCI and Sarbanes-Oxley (SOX) compliance.
- A common mitigation for several regulatory compliance programs is a specific set of scans against a specified set of vulnerabilities.

Scanning

- Vulnerabilities are special forms of errors, in that they can be exploited by an adversary to achieve an unauthorized result.
- As in all other types of defects, vulnerabilities can range in severity, and this is measured by the potential impact on the overall system.
- Scanning for known vulnerabilities is important as attackers will do this very thing on code bases.
- Vulnerabilities are frequently found during activities such as penetration testing and fuzz testing.
- The nature of these testing environments and the types of results make vulnerability discovery their target of opportunity.
- By definition, these types of errors are more potentially damaging, and they will score higher on bug bar criteria than many other error types.

SP 2025

Simulations

Simulations

- Simulation testing involves testing the application in an environment that mirrors the associated production environment.
- Examining issues such as configuration issues and how they affect the program outcome is important.
- Data issues that can result in programmatic instability can also be investigated in the simulated environment.
- Simple applications may have simple setups, but complex applications can have significant setup issues.
- Simulation testing can go a long way toward discovering issues associated with the instantiation of an application and its operation in the production environment.

Simulation

- Simulation testing can provide that last testing line of defense to ensure the system is properly functioning prior to deployment.
- This is an opportunity to verify that the interface with the operating system is correct and that roles are properly configured to support access and authorization.
- It also checks that firewall rules (or other enforcement points) between tiers/environments are properly documented, configured, and tested to ensure that attack surface/exposure is managed.
- Other benefits of simulation testing include validating that the system itself can stand up to the rigors of production performance

SP 2025

Failure Modes

Failure Modes

- Neither all errors in code result in failure, nor all vulnerabilities are exploitable.
- During the testing cycle, it is important to identify errors and defects, even those that do not cause a failure.
- Although a specific error, say one in dead code that is never executed, may not cause a failure in the current version, this same error may become active in a later version and result in a failure.
- Leaving an error such as this alone or leaving it for future regression testing is a practice that can cause errors to get into production code.
- Fault Testing and Stress Testing are used in this phase.

Failure Modes

- Fault testing (break testing) is a specific test that uses faults to ensure they generate errors and the errors are properly handled.
  - Break testing is where one uses inputs that are specifically designed to trigger failures.
  - Then the handling of these failures is measured to ensure the software can still function.
  - Although most testing is for failure, it is equally important to test for conditions that result in incorrect values, even if they do not result in failure.
  - Incorrect values have resulted in the loss of more than one spacecraft in flight, even though the failure did not cause the program to fail, it did result in system failure.
- Stress testing is the use of heavy loads on software to ensure that it still functions reliably, both with use and misuse cases.
  - A common failure condition is load testing, where the software is tested for capacity issues.
  - Understanding how the software functions under heavy load conditions can reveal memory issues and other scalerelated issues.
  - These elements can cause failure in the field, and thus extensive testing for these types of known software issues is best conducted early in the development process where issues can be addressed prior to release.

SP 2025

Cryptographic Validation

Cryptographic Validation

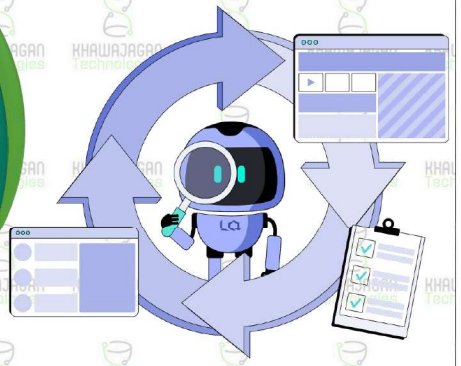
- Having secure cryptography seems easy i.e. use approved algorithms and implement them correctly and securely.
- Cryptographic errors come from several common causes based on above two issues. e.g.
  - Choosing to develop custom/own cryptographic algorithm is a wrong decision as developing a secure cryptographic algorithm is far from an easy task, and even when done by experts, weaknesses can occur that make them unusable.
    - Deciding to use a trusted algorithm is a proper start.
  - Instantiating the algorithm properly is another hurdle in this regard.
    - An easy way to avoid this type of error is to use a library function that has already been properly tested.
    - Sources of these library functions abound and provide an economical solution to this functionality's needs.
  - The generation of a real random number is not a trivial task as computers are machines that are renowned for reproducing the same output when given the same input.
    - There are functions for producing random numbers built into the libraries of most programming languages, but these are pseudo-random number generators, and although the distribution of output numbers appears random, it generates a reproducible sequence.

Cryptographic Validation

- Having secure cryptography seems easy i.e. use approved algorithms and implement them correctly and securely.
- Cryptographic errors come from several common causes based on above two issues. e.g.
  - Entropy is a measure of the randomness of a bit sequence, and true random numbers have high entropy. High entropy sequences can also be an indication that the value is a random number, making the storing of a random number in memory discoverable to an attacker.
    - Random numbers must be handled in special and specific ways when in memory and stored to prevent their discovery and recovery by an attacker.
  - Protecting the keys and the seed values and ensuring proper operational conditions are met have proven to be challenging in many cases.
    - The bottom line is simple: do not hard-code secret keys in your code.
    - Keys should be generated and then passed by reference, minimizing the travel of copies across a network or application.
      - Storing them in memory in a noncontiguous fashion is also important to prevent external detection.

SP 2025

Regression Testing



Regression Testing

- As changes to software code bases occur, they must be tested against functional and nonfunctional requirements.
- This is the normal testing that occurs as part of any change process.
- Regression testing is the testing of the changes when applied to older versions of a code base.
- This is common in large software projects that have multiple versions distributed across a customer base.
- The challenge is not in the direct effects of a change, but in interactive changes that occur because of other code differences between the two versions of a program.
- Regression testing can be expensive and time-consuming and is one of the major challenges for a software vendor that is supporting multiple versions of a product.

Regression Testing

- Regression testing is not as simple as completely retesting everything—this would be too costly and inefficient.
- Depending upon the scope and nature of the change, an appropriate regression test plan needs to be crafted.
- Simple changes to a unit may require a level of testing be applied only to the unit, making regression testing fairly simple.
- In other cases, regression testing can have a far-reaching impact across multiple modules and use cases.
- A key aspect of the patching process is determining the correct level, breadth, and scope of regression testing that is required to cover the patch.

SP 2025

Integration Testing

Integration Testing

- Even if each unit tests properly per the requirements and specifications, a system is built up of many units that work together to achieve a business objective.
- There are emergent properties that occur in systems, and integration (or systems-level) testing should be designed to verify that the correct form and level of the emergent properties exist in the system.
- A system can be more than just the sum of the parts, and if part of the "more" involves security checks, these need to be verified.
- Systems or integration testing is needed to ensure that the overall system is compliant with the system-level requirements.
- It is possible for one module to be correct and another module to also be correct but for the two modules to be incompatible, causing errors when connected.
- System tests need to ensure that the integration of components occurs as designed and that data transfers between components are secure and proper.

SP 2025

Continuous Testing

Continuous Testing

- Continuous testing is the use of automated testing as part of the software delivery process.
- This is done to collect data on the business risk issues associated with a software release candidate in a rapid form, allowing the development team fast access to the results.
- Continuous testing is embedded within the development process, not tacked on at the end, and its feedback is an integral part of the development process.
- Continuous testing requires the team to continuously review the test suite to eliminate redundancy and optimize risk reporting.


WHY?
WHAT?
WHERE?
WHEN?
HOW?
WHO?
WHICH?
WHOSE?
KHAWAJAGAN
Technologies