Assuring Software Security Through Testing

White, Black and Somewhere in Between

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA

Introduction

Take any software development project plan today and it is more than likely that the plan will not have a line item with time allocated exclusively for security testing. It is only a matter of time before software deployed or released without attestation of its ability to withstand attacks will be hacked. It is not a question of if the software will be hacked, but when it will be hacked.

(ISC)²®'s whitepaper, *Code (In)Security,* highlights various considerations that need to be taken into account to develop code that is secure. But merely developing secure code without attesting to its assurance capabilities is akin to operating an automobile without checking to ensure that the brakes work as expected. With such an outlook, a crash becomes not just possible but inevitable. This paper will discuss the need for attesting software assurance, the different types of testing as it pertains to functionality and assurance, a security tester's profile, and some proven strategies to incorporate security testing into the software development lifecycle (SDLC).

The Need For Security Testing

Before we dive into discussing the need for security testing, it is important first to recognize that there are three key quality components to software assurance, as illustrated in Figure 1: reliability, resiliency, and recoverability. Reliable software is that which functions as needed by the end user. Resilient software is that which is able to withstand the attempts of an attacker to compromise confidentiality, and/or impact integrity, or availability (CIA). Finally, recoverable software is software that is capable of restoring itself or being restored to expected normal operations when it has failed in its reliability or resiliency.

Figure 1. Software Quality Components

Most commonly, when software is said to be of "quality," it essentially means that the software is working as designed and expected. This is primarily a consideration of software functionality, and not its assurance capabilities. Today, however, besides the reliability aspect of software quality, it is also imperative to take into account the security of the software. This two-pronged approach to software quality testing ensures that software is not only reliable but resilient to withstand attacks that impact CIA.

Security testing is necessary because it has a distinct relationship with software quality. Just because software meets quality requirements related to functionality and performance, it does not necessary mean that the software is secure. The inverse however is true: i.e. software that is secure is software with added resiliency, thus software of higher quality. For example, when the "Add to cart" button on a web page is clicked and the selected product is added to the cart (functionality) in less than the expected two-second requirement (performance), it can be said to meet the reliability quality requirements as established by the

business. But if the software is not tested for security, there is no guarantee that the product code that is added to the cart has not been tampered by an unauthorized user. Poor architecture and implementation of the web application cannot assure the CIA aspect of software assurance, which would otherwise indicate the resiliency quality of the software.

Types Of Software Testing

In the following section, we will cover the various types of software testing. These tests are categorized into reliability testing, recoverability testing, and resiliency testing.

Reliability Testing

Reliability testing is primarily performed to attest to the functionality of the software and is essentially the reason for having a testing phase in the SDLC. It ensures that the software functions as is expected by the business. It is also commonly referred to as functional testing. Validation of software functionality can be accomplished using any one or a combination of the following tests.

Unit Testing: The most fundamental method to validate the functionality of the software is by breaking the software functionality into smaller parts and testing each unit in isolation. Unit testing, unlike other testing methods, is not performed during the testing phase of the SDLC but in the implementation (coding) phase. It is performed by the developers. Unit testing provides an opportunity to catch functional, logic, and security bugs early on in the SDLC. The logic behind unit testing is that it is easier to find the needle in the haystack when the haystack is broken down into more manageable pieces.

Software architecture plays a major role in the ease and effectiveness of unit testing. Software that is designed with modular programming concepts such as high cohesion (discrete functionality in modules) and loose coupling (dependencies between units) can be more easily unit tested than software which is not.

Unit testing also gives insight into Quality of Code (QoC) issues because when one tests the source code – line by line, unit by unit – inefficiencies, circular dependencies, and vulnerabilities can be uncovered. Code inefficiencies include remnants of code which serve no required functionality, and infinite loop constructs that exhaust hardware resources. Developers sometimes implement complex business logic in not a very linearly-independent manner, but rather in a cyclomatically, complex manner with circular dependencies, which not only violates the secure design principle of economy of mechanisms, but also impacts the performance of the software. Unit testing is useful to help uncover such complexities, besides helping to discover common coding vulnerabilities, such as hard coding values, and sensitive information such as passwords and cryptographic keys in the code itself (inline).

Additionally, unit testing facilitates collective code ownership in agile development methodologies, such as extreme programming (XP) or Scrum. With accelerated development efforts and the entire software team collectively responsible for the code that is released, unit testing can help in identifying any potential issues made by a programmer on the shared code base before it is released. It extends the test coverage and, when integrated with automated build scripts and tools, it can be used to automate the testing process.

Logic Testing: The primary purpose of logic testing is to validate the accuracy of the software's processing logic. It is particularly necessary to validate the implementation details of code that is copied from other modules or code that is determined as cyclomatically complex. Additionally, logic testing must not be ignored in situations where the logic of the software is dependent on user input.

Logic testing includes testing preconditions (if-then-else), and looping constructs (for, for each, do while, while, etc.). It also includes testing using predicate – something that is affirmed or denied based on the logic. Boolean predicates return a true or false depending on whether the logic condition was met. Operators such as "AND", "OR", "NOT EQUAL TO", "EQUAL TO", etc., are used to vary or negate intended functionality when conducting logic testing. Blind SQL injection testing is a form of testing using Boolean predicates, where the attacker uses true or false probing queries to discover internal database architecture and implementation.

"Blind SQL injection testing is a form of testing using Boolean predicates"

Integration Testing: The logical extension to unit testing is integration testing which is testing the software after the units are integrated into a whole. In integration testing, the functionality of the sum of all parts is validated. While unit testing results could indicate that the software functionality of the individual components is working as designed and expected, integration testing provides insight into how the system will function and perform as a whole. Additionally, when software is developed in a distributed manner (multiple developers, multiple locations, etc.), integration testing is imperative to ensure the reliability of software operations.

Regression Testing: Business requirements change over time and newer functionality is often added to software code. Whenever code is modified there is the potential for breaking existing functionality. This is where regression testing comes in handy. Regression testing is performed before the software is released or deployed to ensure that the software does not regress to a non-functional or insecure state. Regression testing is sometimes referred to as verification testing.

Regression testing is performed mainly on coding issues over design flaws. It is particularly important for bug fixes as there is a tendency for a software patch to break some other functionality outside the scope of the fix. It is performed to ensure that it is not just the symptoms but the root cause of the issue that is fixed, besides verifying that the bug fix did not introduce any new bugs or cause previously fixed bugs to reappear. Additionally, regression testing must be performed whenever the data is changed or the database is modified (changed column names, data types, size, etc.), since these changes can potentially have side effects, reverting functionality or reducing the security of the software.

It is important to allocate time in the project plan for regression testing. A best practice is to predefine and use a library of tests

"It is important to allocate time in the project plan for regression testing."

prior to the release of any version. However, one must recognize that having a predefined set of tests may not cover newer threats to software assurance.

Simulation Testing: When software that functions without any problems in the development environment experiences hiccups in the production environments, it is indicative of issues with configuration management. This means that the development and production environments are disparate in their configurations. Since it is not advisable to conduct software testing in the production environment, a mirrored or simulated environment needs to be created and the software should be tested in this simulated environment. Because the configuration settings of the production (operational) environment are simulated, any issues of the software not working can be determined early and addressed prior to deployment.

Recoverability Testing

Recoverability testing is primarily performed to attest to the ability of the software to restore itself or be restored to normal business operations. It can also be referred to as performance testing. The two most common types of performance tests include load testing and stress testing, and these address the availability tenet of the CIA security triad.

Load Testing: The goal of load testing is to determine the maximum operating capacity for the software by subjecting the software to a large volume of tasks, users, or data. In layman's terms, it is subjecting the software to duress with the goal of identifying its breaking point. The strategy employed with load testing is that the tests are iterative in nature. We start with a small volume and then move in increments, until the peak load up to which the software operates as expected is determined.

Stress Testing: Stress testing complements load testing. Once the peak load is known, stress testing goes one step further to determine how the software will respond when that peak load is exceeded. It is subjecting the software to loads beyond its breaking point to determine how the software will react. In addition to providing insight into the recoverability of the software, stress testing also gives insight into the fail secure (or fail safe) capabilities of the software. Stress testing can also help identify resource exhaustion, timing synchronization, and leakage issues.

It must be recognized that the implementation of security features in the software can potentially impact the performance of the software and so it is imperative to conduct performance tests to determine if the software will meet the service level requirements as expected by the business.

Resiliency Testing

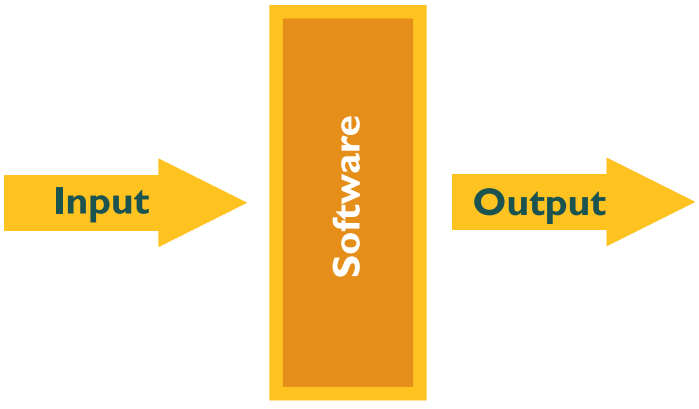
Resiliency testing has to do with the attestation of the ability of the software to withstand attacks. In other words, it is security testing. It ensures that the software is designed and developed with security controls in place that mitigate the risk of exploitation. In this section, common approaches (processes) and techniques of security testing will be covered.

Approach

There are different approaches to testing the software to ensure its hack-resilience. The two main approaches are black box testing and white box testing.

Black Box Testing: Otherwise known as zero-knowledge testing, black box testing considers the software to be a black box as depicted in Figure 2. The testers have no knowledge about the software architecture or how it is implemented. They take a hostile user's perspective and test the software by checking how the software behaves by passing in inputs and observing the output. It is, therefore, primarily a behavioral analysis of the software.

Figure 2. Black Box Testing



White Box Testing: Unlike black box testing, with white box testing the testers have substantial knowledge of the software, ranging from how it is designed to, and in most cases, even the source code. The analysis is very structured in nature and, depending on the availability of time and resource, can include the entire code base and related data flow analysis. When the source code is analyzed, security vulnerabilities are more likely to be detected, in addition to insider threats posed by seemingly innocuous maintenance hooks that can posed by implanted logic bombs or seemingly innocuous maintenance hooks that can serve as a back door to attackers.

Although it may seem that in order to get an accurate picture of the vulnerabilities of the software, white box testing would be the preferred choice since it provides more extensive code coverage than black box testing, this may not always be true. Just because code compiles correctly without error does not necessarily mean that it will run (execute) without any security problems. Denial of service issues caused by resource exhaustion and deadlocks are less likely to be detected in code review (a white box kind of test), and black box testing is more useful in this endeavor. Table 1 depicts a summarized set of criteria and the testing approach that can be used to address each. In reality it is a hybrid approach, also known as gray box testing, that is recommended and often employed to attest to the presence and effectiveness of the security controls in the software.

Techniques

In addition to understanding the two common approaches to security testing, the attestation of the software's ability to withstand attacks can be achieved by vulnerability assessment and/or penetration testing. Although vulnerability assessment and penetration testing are often synonymously used, there is a distinction between the two.

Vulnerability assessment aims at identifying weaknesses in software which are indicative of the absence of security controls that can potentially mitigate the vulnerability. When security testing involves vulnerability assessment, the team is looking to see if the security controls that were identified as required were indeed designed and implemented during the architecture and development phase of the SDLC. These controls can include encryption, hashing, masking, and input validation. Vulnerability assessments are usually done using the white box testing approach.

Penetration testing includes performing a vulnerability assessment but does not merely verify the presence or absence of security controls. Penetration testing goes one step further by evaluating and validating the effectiveness of the security controls that have been designed and implemented. When conducting a penetration test, the security tester plays the role of a malicious hacker and attempts to break through (penetrate) the software controls,

Table 1 – Comparison between Black Box and White Box Security Testing

Criterea	Security Testing Approach	
	Black Box	**White Box**
Determination of root cause	Most likely to address the symptoms than the root cause.	Exact line of code or design issue causing the vulnerability can be identified.
Extent of code coverage	Limited as the analysis is behavioral; not all code paths may be covered.	Greater as the source code and configuration is available for review.
Detection of logic flaws	Not knowing the normal behavior of the software; anomalous behavior may not necessarily indicate flaws in logic.	The availability of design and architectural documents besides code can be used to detect logic flaws.
Issues with deployment	Assessment can be performed in pre- as well as post-deployment production or production-like simulated environment, giving insight into any potential issues after deployment.	Assessment is performed in pre-deployment environments usually providing limited issues pertaining to configuration and change management.

thus compromising the confidentiality, integrity, and/or availability aspects of software assurance. When a penetration test is conducted, it is important to recognize that the appropriate scope and rules of engagement are established because the outcome of a test can be disruptive to the business, especially if the test includes active exploitation of detected vulnerabilities. Penetration tests usually begin with reconnaissance activities such as scanning and spidering to gaining root privileges on the target system. Because penetration tests take a hostile user's perspective, they are usually conducted using the black box testing approach, although this may not always be the case.

One of the most prevalent mechanism used to attest to the strength and effectiveness of security controls, more particularly input validation, is fuzzing. By passing random to pseudo-random data (called fuzz) into the software application and observing

"Fuzzing is also commonly referred to as fault injection testing."

how the software handles the fuzz data (also known as fuzz oracle), security bugs can be detected. Fuzzing is also commonly referred to as fault injection testing. In addition to software data, network protocols, application programming interfaces (APIs), and file formats can be fuzz tested. Though fuzzing may seem to always be a black box testing type of activity, it is not exclusive to that type of security testing. A fuzz test can be a white box assessment when the format (specification) of the input (data) that the software expects is known in advance. This type of fuzzing is referred to as "smart" fuzzing. Conversely, when the fuzz that is supplied as part of the attestation does not take into account the expected format of the input, it is referred to as "dumb" fuzzing. Smart fuzzing is obviously the preferred option as the results of dumb fuzzing can be relatively more disruptive to the business. In addition to validating the effectiveness of input validation controls, fuzzing also gives insight into the exception and error handling capabilities of the software.

Profile

One of the most important and crucial elements of a good security test is the very people conducting the tests. In the area of security testing, as with any other aspect of software assurance, (ISC)²'s mantra: "It's the people; security transcends technology," holds true.

What makes a good security tester?

Security testers are a breed all their own. In addition to an anti-developer attitude, a good security tester's profile includes a creative mind to think out of the box. A good security tester thrives and excels when posed with challenges. They have a

"Good security tester's profile includes a creative mind to think out of the box."

constant thirst for learning newer attack techniques and coming up with all possible combinations of attack as they seek to exploit weaknesses in the software or system. They usually have self-driven personalities and their motivations are often related to ego than materialistic pursuits. With appropriate training and skills on software security throughout the SDLC, these testers become very valuable to the organization.

Strategies

Since security testing is not an optional activity, it is imperative to incorporate this vital activity into the SDLC. Security testing should be made an integral part of the overall testing process. It must be included into the scope of the software development

"Security testing should be made an integral part of the overall testing process."

project, even before the code for that software is written. A proven strategy is to incorporate a library of security tests into the enterprise project template (assuming you already have one) and maintain these tests in the centralized test and bug tracking database/software. This will ensure that all projects will, at a bare minimum, automatically inherit any tests that need to be run as part of the project.

Another strategy is to start generating the test cases during the requirement phase of the SDLC itself. When the requirements are known, there is no need to wait until code is complete to generate all the test cases. This proactive approach to testing gives time for both functional and security test cases to be generated in advance and the testing phase can be used efficiently to conduct these test cases.

Additionally, training the software testers to conduct security testing has its benefits. It not only ensures the resiliency (security) aspect of software in addition to its reliability and recoverability (functional) aspect, but it also empowers the software testers and augments their marketability and career prospects.

Conclusion

Software assurance is comprised of reliability, recoverability, and resiliency aspects of the software. Software testing must address all of these. Without testing the software for its security capabilities, it is only a matter of time before software will be exploited (hacked). Software testing for functionality should always be augmented with security testing for resiliency. Security is an attribute of quality, and software that is less prone to getting hacked can be said to be of higher quality than software that doesn't take security into account.

Unit testing is performed by developers and has the benefit of detecting functional and software assurance issues early on in the lifecycle because it breaks the software into small manageable units. Logic testing is useful to ensure that the software's processing logic is accurate and as expected by the business. Integration testing is useful to ensure that the system will function as a whole when the individual components (units) are brought together. Regression testing is necessary when code changes. It can be used to compute the relative attack surface from one version to another and provide insight into whether the state of software security is improving or deteriorating. Simulation testing is useful to detect data and configuration mismatches between non-production and production environments.

Performance testing includes load testing and stress testing. These are used to determine bottlenecks and to identify maximum thresholds up to which the software can optimally perform as expected by the business. Performance testing is useful to determine issues pertaining to resource exhaustion and race conditions.

The two most common approaches to security testing are black box testing and white box testing. In reality however, gray box testing – a combination of both black box and white box testing – is the approach more often used. Vulnerability assessments that verify the presence of security controls, and penetration testing which is used to determine if those security controls are effectively working, are common security testing techniques. Another prevalent technique is fuzzing which uses fault injection to check the effectiveness of input validation and error handling mechanisms.

Security testers play a vital role in attesting to the resiliency of software. Incorporating a library of security tests into the enterprise project templates, proactively generating test cases early on in the SDLC, and educating and empowering software testers to also take into account security testing are proven strategies to incorporate security testing into the SDLC.

In conclusion, it is important to recognize that the color of the pieces in a game of chess provides no advantage. How the game is played is what makes the difference. Similarly, how the organization uses black box or white box testing is what makes the difference in whether the software is secure or not. Security testing can be used to achieve software assurance – akin to setting up the chess board so that you are able to defend your organization and avoid checkmate.

About (ISC)²®

(ISC)² is the largest not-for-profit membership body of certified information security professionals worldwide, with over 70,000 members in more than 135 countries. Globally recognized as the Gold Standard, (ISC)² issues the Certified Information Systems Security Professional (CISSP®) and related concentrations, as well as the Certified Secure Software Lifecycle Professional (CSSLP®), Certified Authorization Professional (CAP®), and Systems Security Certified Practitioner (SSCP®) credentials to qualifying candidates. (ISC)²'s certifications are among the first information technology credentials to meet the stringent requirements of ANSI/ISO/IEC Standard 17024, a global benchmark for assessing and certifying personnel. (ISC)² also offers education programs and services based on its CBK®, a compendium of information security topics. More information is available at **www.isc2.org.**

About the Author

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA is CEO and President of Express Certifications and SecuRisk Solutions, companies specializing in professional training, certification, security products and security consulting. His security experience includes designing and developing software security programs from Compliance-to-Coding, application security risk management, security strategy and management, and conducting security awareness sessions, training, and other educational activities. He is currently authoring the Official (ISC)² Guide to the CSSLP, is a contributing author for the Information Security Management Handbook, writes periodically for Certification, Software Development and Security magazines and has contributed to several security topics for the Microsoft Solutions Developer Network. He has been featured in various domestic and international security conferences and is an invited speaker and panelist in the CSI (Computer Security Institute), Catalyst (Burton Group), TRISC (Texas Regional Infrastructure Security Conference), SC World Congress, and the OWASP (Open Web Application Security Project) application security conferences. He can be reached at **mano.paul@expresscertifications.com** or **mano.paul@securisksolutions.com.**

References

Brown, Jeremy. "Fuzzing for Fun and Profit." Krakow Labs Literature, 02 Nov. 2009. Web.

Gallagher, Tom, Bryan Jeffries, and Lawrence Landauer. Hunting Security Bugs. Redmond: Microsoft, 2006. Print.

Kelley, Diana. "Black Box and White Box Testing: Which Is Best?" Search Security.com. 18 Nov. 2009. Web.

"OWASP Testing Guide V3." OWASP. 15 Sept. 2008. Web.

"Penetration Testing vs Vulnerability Assessment | Darknet - The Darkside." Darknet - The Darkside | Ethical Hacking, Penetration Testing & Computer Security. 25 Apr. 2006. Web. 17 Aug. 2010.