Develop a Security Testing Plan for the Hackazon Application

Hayden Eubanks

School of Business, Liberty University

CSIS 485-B01

Prof. Backherms

October 9, 2023

Develop a Security Testing Plan for the Hackazon Application

Introduction:

In the modern landscape of technological interactions, one aspect of special concern is application security and the protection of system data and resources. The lack of security could be seen to have major implications on the hosting organization and as such strong security must be expressed in the final product. In order to achieve this security, security-centric design must be implemented to integrate security throughout the design of a software product (Tsui, Karam, Bernal, 2016). Security-centric design can then be seen to include the area of system testing and through adequate testing, a system can be proven as secure while vulnerabilities are identified for correction before they are exploited by a malicious actor (Merkow & Raghavan, 2012). However, the testing process itself can vary in quality, and with this, inadequate testing could result in vulnerabilities or bugs being missed and making it into production of the final product. Additionally, the lack of adequate testing could further perpetuate failures in quality as testing is also used to verify that system requirements are fulfilled in a given implementation (NCSC, 2019). From this, the importance of the testing phase of the software development lifecycle (SDLC) can be seen, and with this, the need for an organized approach to application testing can be identified. The need for a systematic approach to security testing has long been recognized in the sphere of software development, and as such several standards have been produced to enable a repeatable approach to generating secure software through testing practices (NIST, 2006). Through examining these standards and best practices, a security professional can be best equipped to implement application security testing and in doing so increase the resiliency of the software products they address.

Examine software testing methodologies:

With the importance of a comprehensive and repeatable approach to software security testing identified, an examination of industry standards and best practices can then be performed to gain an increased understanding of effective testing methodologies. One of the most used standards within the field of security testing today is the OWASP Application Security Verification Standard (ASVS) and this standard can be seen to provide guidance to the security testing process allowing security professionals to implement comprehensive security testing throughout a system (OWASP, 2021). One of the core objectives of the ASVS is to empower security testers to integrate security testing throughout the development of a software product and in doing so define a strong methodology for testing an application (OWASP, 2021). A security professional is then able to utilize a testing methodology outlined by a standard such as the ASVS to test the security of their applications and ensure their products are secure and relevant to the domain they will be implemented within.

To understand the areas of testing addressed within the ASVS, an understanding of the ASVS architecture and the levels of security addressed within the ASVS guidance must be understood. The OWASP organization identifies three levels of security that can be achieved by a software product (OWASP, 2020), and understanding the level of security provided by each level can allow a security professional to select an appropriate security level for their application’s use case. The first of these levels, level 1, is the most basic security level and the security provided by this level could be observed as sufficient for simple applications with minimal security implications (OWASP, 2021). Security concerns defined within this testing level can be primarily automated as a low level of dynamic interaction from the user is needed to affirm this security level has been achieved. The next level, level 2, can then be seen as the level of security recommended for most web applications whose security does not directly affect the well-being or safety of users (Merkow & Raghavan, 2012). Unlike the first level, the second level integrates much more manual testing to detect vulnerabilities that may be difficult to detect by automated means (Shahid et al., 2022). With this, the third level can then be seen to express the highest achievable level of security and is a relevant security goal for applications dealing with highly sensitive or critical data whose compromising could result in actual harm to individuals (OWASP, 2021). This level requires the most comprehensive approach to security testing and as such demands rigorous automated and manual inspection of all system components to ensure security exists within all sublevels of the software (Ksiezopolski et al., 2022). Further, the third testing level thoroughly examines the design process itself to verify that this process is robust and will produce a secure application as a result (Merkow & Raghavan, 2012). As the security levels defined in the ASVS increase, both the security requirements and effort needed to achieve the security level also increase proportionally highlighting a multidimensional expansion in project depth and breadth (Merkow & Raghavan, 2012). This then highlights the relationship between input testing effort and system security and from this, an evaluation of the tradeoff between implementation costs and security achieved can be performed by the product owners to ensure the correct level of resources are allocated to achieve the most relevant level of security.

With an understanding of the security levels involved in the ASVS achieved, an examination of the areas of security addressed throughout the standard can then be performed. Throughout the ASVS, several testing domains are identified including testing that can be performed through manual inspection, automated processes, or the use of specialized tools specific to an application (Ksiezopolski et al., 2022). Each of these testing types provides value to the testing process and through integrating each of these methodologies into software testing, a greater degree of security can be achieved (Tsui, Karam, Bernal, 2016). In addition to this, the ASVS defines four areas of testing that can greatly contribute to the secure production of resilient software. These testing areas include manual source code review, automated source code analysis, automated dynamic analysis, and penetration testing and the combination of each of these testing areas can then be seen to address every level of software design (OWASP, 2021). The first of these testing areas, the manual inspection of source code involves security testing on the lowest level of abstraction where the tester is able to examine the security features of the source code itself. This can assist the tester in identifying code errors as well as vulnerabilities related to deprecated components or insufficient implementations (Merkow & Raghavan, 2012). For example, the manual inspection of the source code could identify a function that has known vulnerabilities associated with it or reveal that an input field does not have validation code attached to it. All code is written on this level of abstraction, and as such the importance of performing system tests on this level can be seen. The next type of testing, automated source code analysis, can then be seen as an extension of this testing format with the primary difference being the automation of source code review tasks. Automating the review of source code can assist the testing process by allowing the easy-to-detect errors to be automatically tested for, freeing up developer resources to manually test more complex elements of the system (NCSC, 2019).

While the two previous testing methods used system knowledge to perform white box testing, the remaining two strategies involve dynamically testing the system without internal knowledge of the underlying system (Merkow & Raghavan, 2012). This is commonly referred to as black box testing and black box testing can be seen to imitate the system’s exposure to threats as they would be encountered in a real-life scenario (Ksiezopolski et al., 2022). The first approach to black box testing can be classified as the automated testing of dynamic system interactions. The forward-facing elements of an application could be attacked from many different angles highlighting the dynamic nature associated with this type of testing (Merkow & Raghavan, 2012). Many of these dynamic tests can be automated such as the inputting of data into entry fields and observing the results and as such, automation can be seen as a valuable tool in dynamic testing. This then leads to the final testing type, penetration testing, which refers to the manual dynamic testing approach performed on the system (Ksiezopolski et al., 2022). During penetration testing a security professional can attempt to expose system vulnerabilities through performing mock attacks on the system. Penetration testing can be seen as an extremely valuable tool in security testing as it most closely models the way that a malicious actor would attempt to attack a system. Each of the testing procedures provides benefits to the testing process and through combining each of these testing methodologies a greater level of security can be realized within a system.

Compare Static and Dynamic Testing Methodologies:

Extending from the analysis of software security testing methodologies, it can be seen that each testing procedure can be broadly classified as a static or dynamic test method (OWASP, 2020). Each of these testing types provides benefits to the security testing process and as such it is important for a security professional to understand and be able to implement tests of both types. Through examining static and dynamic testing, a security professional can then be equipped to engage with all forms of security testing and increase the security of their software systems.

The first form of testing, static testing, refers to the testing of static system components such as the source code as the source code will remain unchanged while interacting with the application (NCSC, 2019). As static testing relates directly to the source code, it is often referred to as white box testing indicating that the testers have knowledge of the internal system workings (Tsui, Karam, Bernal, 2016). Static testing can be an extremely valuable level by which to perform testing as it relates directly to the level at which code is written. This means that if an error exists due to the way that a section of code is written, it can be detected and corrected within that section of code providing efficient security optimization (OWASP, 2020). Additionally, static analysis can allow for the detection of deprecated functions or implementations that leave the system vulnerable to exploitation (Merkow & Raghavan, 2012). These types of errors can be difficult to detect without knowledge of the inner workings of a system and as such, static testing can be seen as extremely valuable in verifying system security through testing. By utilizing a combination of automated and manual source code reviews, security testers can increase their ability to identify errors in code and increase the security of a system.

The next form of system testing, dynamic testing, can then be seen as the testing of system infrastructure that requires access to non-static system components such as ports or dynamic user actions (NCSC, 2019). These types of tests are regularly implemented without knowledge of the underlying system and as such are frequently labeled as black box tests (OWASP, 2021). Dynamic tests involve factors of the system that could change during interaction such as the ports through which communications occur and through testing these dynamic system elements a greater understanding of operational security can be gained (NCSC, 2019). This form of testing can then be seen to operate on a higher level of abstraction than the previously mentioned static tests as the testing is performed on the system interface rather than the source code itself. As the interface is the element of the system that users will interact with, it is essential that thorough testing be carried out to ensure that adequate security is implemented throughout the interface. This highlights the importance of dynamic testing and automated testing can be performed in conjunction with manual penetration testing to ensure the interface has mitigated the vulnerabilities it faces (Merkow & Raghavan, 2012). Combining static and dynamic testing can then be seen to create a layered approach to system testing and through implementing a layered approach, security can be increased on all levels of design (Tsui, Karam, Bernal, 2016). For this reason, security professionals must understand and be ready to implement both static and dynamic tests on their software systems.

Develop a Security Test Plan:

Having gained an understanding of the testing strategies a security professional can use to increase system security; it is then possible to develop a security testing plan. The security testing plan outlined in this document will detail a security testing plan for the Hackazon application and will address security testing methodologies such as the manual and automated reviewing of source code, automated dynamic testing, and manual penetration testing. Reviewing each of these testing domains within the context of the Hackazon application can then outline a plan for testing that can guide testers through comprehensive testing procedures. Planning is an important aspect of the testing process, and through generating a testing plan security testing can be more thoroughly conducted and the system security increased as a result (NCSC, 2019). Additionally, system testers will be able to reference the system test plan to compare their progress in testing and identify the remaining tests and resources that need to be allocated to ensure sufficient security is implemented within the system (Merkow & Raghavan, 2012). Security testing is a core element of the SDLC and through organizing this phase of development, a greater level of quality and security can be observed in the final product.

The first testing strategy that will be performed on the Hackazon application is the manual review of source code by a security tester. Several elements of system security can be difficult to detect in the source code by automated means and as such should be manually inspected by a system tester (OWASP, 2021). One way that this can be addressed within the Hackazon application, is through the thorough examination of the structure of the code to identify any structural design failures. The poor structuring of a code could lead to unintended consequences during execution and in the worst case could open up the system to security vulnerabilities (Tsui, Karam, Bernal, 2016). Additionally, while a section of code may function correctly, it may be implemented with deprecated methods that inherently present security risks to the system's implementation. For example, the use of deprecated encryption functions could result in the encryption being easily broken revealing the values of confidential data (Basta, 2018). The manual inspection of code could then further reveal the lack of security requirement implementations or technical controls that result in the system lacking in security. Identifying these failures in implementation is crucial to the resiliency of a software product highlighting the importance of manual code inspection in software testing.

The next area of system testing to be implemented within the Hackazon application can then be seen as the automated testing of source code values. One specific area of the source code that can be tested through automated means is the inclusion of input validation at all aspects of the code that receive user input. In the improved security implementation for the Hackazon application, input validation will be performed by dedicated validation functions, and through scanning the source code, the verification of these functions' existence can be obtained. Another area in which the automated scanning of source code could be used to test security is through scanning the code for the existence of implemented methods with known vulnerabilities associated with them. This view can then be used to allow a developer to quickly update deprecated functions when vulnerabilities are discovered ensuring the timely security of the application. Automation is a powerful tool in development as it can free testers to devote their time to more complicated tasks (NCSC, 2019), and as such the automation of Hackazon's source code will be implemented wherever possible.

Following this, automated test cases can be generated to test the application's response to dynamic test elements. Within the context of the Hackazon application, dynamic tests can be set up to try connecting to the server through several ports to ensure only validated ports are used for connection. Further, test cases can be created to attempt to access system resources through automated URL manipulation in which the test will return if the resources were accessed by this method. Another area in which automated dynamic testing can be performed can then be seen through the use of scripts to simulate user usage and specifically to repeat the steps that have been known to generate a vulnerability. From this, the importance of identifying repeatable steps to generate a bug can be seen (OWASP, 2021) and these steps can be used to create scripts that validate when a security bug has been corrected in the software. Finally, automation can be utilized in dynamic system testing to express test cases such as the validation of input fields. Each of the input fields could be fed an invalid entry for that field to test the way that the system responds to invalid data. This process can be fully automated and a report generated for the security professional to examine any fields that are prone to incorrect data entry of injection attacks (NCSC, 2019). From this, the power of automation in dynamic testing can be seen and through implementing these tests, the security of the system can be verified whenever the automated test cases are run creating a repeatable approach to security testing (Tsui, Karam, Bernal, 2016).

The final test strategy to be implemented within the Hackazon application can then be seen as manual penetration testing by a security professional. Penetration testing most closely resembles the way that a malicious actor would interact with the application (OWASP, 2020) and as such provides an incredibly valuable lens through which to observe system security. One area of the Hackazon application that could greatly benefit from penetration testing is manual testing relating to session hijacking attacks. In previous examinations, the application could be seen to lack appropriate encryption to hide key values such as the session ID allowing a malicious attacker to hijack a session. Manual penetration testing could then be used to test a new security implementation and ensure that this vulnerability has been mitigated. Another area by which penetration testing can be used to improve the security of the Hackazon application is the testing of additional attack vectors such as attempting injection attacks, cross-site scripting attacks, or server-side request forgery attacks (OWASP, 2020). In previous examinations, the Hackazon application has proven vulnerable to these forms of attack, and manual penetration testing could be implemented to verify that these vulnerabilities have been adequately covered through further implementations. Penetration testing gives the security professional an understanding of the access a malicious actor could gain to the system and as such can be a valuable metric in determining when a security implementation is sufficient for the organization's security goals (Merkow & Raghavan, 2012).

Through examining each of these testing methodologies within the context of the Hackazon application, it can be seen that the combined security testing approach could be implemented to cover several areas of application testing. This follows the best practice of security-centric design as security can be tested throughout the system's implementation. Further, each test domain provides a different view of application security and allows a security professional to examine each of the views for a more comprehensive picture of the state of the application's security (OWASP, 2020). Planning the security testing phase can ensure that comprehensive testing is performed and in doing so increase the value of the artifacts generated regarding the state of system security (Merkow & Raghavan, 2012). This highlights the value in generating a thorough security testing plan and with this, a security testing plan can be set for the Hackazon application to ensure each area of security is addressed and tested to a sufficient level.

References

Basta, A. (2018). *Oriyano, cryptography: Infosec pro guide.* McGraw-Hill Education. https://bookshelf.vitalsource.com/reader/books/9781307297003/pageid/14

Ksiezopolski, B., Mazur, K., Miskiewicz, M., & Rusinek, D. (2022). Teaching a hands-on CTF-based web application security course. *Electronics (Basel), 11*(21), 3517. https://doi.org/10.3390/electronics11213517

Merkow, M. S., & Raghavan, L. (2012). *Secure and resilient software: Requirements, test cases, and testing methods (1st ed.)*. CRC Press. https://doi.org/10.1201/b11317

NCSC. (February 20, 2019). *Secure development and deployment guidance*. National Cyber Security Centre. https://www.ncsc.gov.uk/collection/developers-collection/principles/continually-test-your-security

NIST. (September, 2006). *NIST Special Publication 800-84*. National Institute of Standards and Technology. https://csrc.nist.gov/glossary/term/test_plan

OWASP. (December 3, 2020). *OWASP web security testing guide (Version 4.2)*. OWASP. https://owasp.org/www-project-web-security-testing-guide/v42/

OWASP. (October 28, 2021). *OWASP application security verification standard (Version 4.0.3)*. https://owasp.org/www-project-application-security-verification-standard/

Shahid, J., Hameed, M. K., Javed, I. T., Qureshi, K. N., Ali, M., & Crespi, N. (2022). A comparative study of web application security parameters: Current trends and future directions. *Applied Sciences*, *12*(8), 4077-4077:23. https://doi.org/10.3390/app12084077

Tsui, F, Karam, O., Bernal, B. (2016). *Essentials of Software Engineering* (4th ed.). Jones & Bartlett Learning. https://libertyonline.vitalsource.com/books/9781284129755