Select Test Cases for the Hackazon Application

Hayden Eubanks

School of Business, Liberty University

CSIS 485-B01

Prof. Backherms

October 1, 2023

Select Test Cases for the Hackazon Application

Introduction:

One aspect that is essential to the development of resilient software is the thorough testing and validation of security requirement implementations. Much of the cost associated with insecure software can be directly attributed to insufficient testing (OWASP, 2020) highlighting the importance of generating sufficient test cases and testing strategies. Further, the software development lifecycle (SDLC) is not limited to technical implementations alone and as such, people, processes, and technology should all be thoroughly tested to mitigate faults that could be introduced in each area of the SDLC (Merkow & Raghavan, 2012). Additionally, software testing must be integrated throughout the development process to ensure that security is sufficient throughout all areas of implementation (NCSC, 2019). This can then be seen to reduce bugs earlier in the SDLC and in doing so reduce the costs and time of implementation (Salva & Regainia, 2019). A thorough understanding of the implications of testing on system security can then allow development teams and security professionals to utilize testing best practices to improve the security of a development project.

This test case report will then seek to address the primary risks facing the Hackazon application by selecting test cases relevant to previously generated system requirements. To accomplish this, an evaluation of the differing levels of system security testing will be performed to give an overview of the different testing areas relevant to the SDLC. Following this, a STRIDE threat model will be created to evaluate and categorize threats making the testing phase regarding these threats easier to visualize and perform (Microsoft, 2022). Additionally, the threat analysis can allow the development team to efficiently allocate resources to areas of testing relevant to risk improving the efficiency of development and resilience of the product deliverable

(OWASP, 2020). Extending from this analysis, the test cases can then be selected for the Hackazon application and modeled in a test case reference table. This table will allow for easy reference during the development process and further facilitates a shared vision of testing that must be implemented within the SDLC (DSIT, 2023). Without guidance, the generation of test cases can be insufficient to verify that security requirements are met within the project (Wang et al., 2022) highlighting the importance of documenting testing requirements and expectations to be met throughout the SDLC.

Compare and Contrast the Different Levels of Security Testing:

As with other areas of the SDLC, security testing is not an isolated activity, but rather layers of activities that are performed toward the common goal of achieving resiliency in software security (Tsui, Karam, Bernal, 2016). These activities that form the sphere of security testing can be generalized into the categories of manual inspection, threat modeling, the reviewing of code, and penetration testing (OWASP, 2020). Each of these areas addresses different elements of the testing process and integrating each of these areas' activities into the SDLC can then be seen to provide a comprehensive and holistic approach to security (NCSC, 2019). Security must be addressed in all areas of design and testing (Wang et al., 2022), and as such it is essential for security professionals to understand each of these testing levels and their roles in the SDLC.

The first level of security testing to be examined is the area concerning the manual review of human elements of design including people, policies, and processes (OWASP, 2020). The most common manner through which these tests are performed can be classified as the analyzing of documentation and the performing of interviews with key individuals of the development process including project stakeholders and developers (Merkow & Raghavan, 2012). These interviews and reviews can provide valuable insight for the development of test cases as they provide a view of the way code or functionality is intended to work, developing a baseline against which testing can be performed (OWASP, 2020). Further, these interviews allow for the testing of the SDLC which is in itself a valuable contributor to the quality of project development (OWASP, 2020).

The next layer of security testing, threat modeling, can then be approached to assess security threats facing a system and prioritize testing resources according to that risk (Microsoft,

2022). This threat analysis builds on the foundations of the risk analysis but furthers that analysis by extending risk analysis to applications (OWASP, 2020). This analysis can assist developers in addressing risks during the development process but additionally provides a framework from which test cases can be generated to test the vulnerability of an implementation to specific risks (NCSC, 2019). This further allows for threats to be categorized and resources allocated in accordance with that categorization to ensure the most prominent risks are the most thoroughly tested for (OWASP, 2020). As the name suggests, the analysis performed in this step can then assist in the modeling of risks into lists and diagrams which can serve as valuable artifacts to the development lifecycle and security testing (OWASP, 2020). Security in software design is aimed at mitigating security threats and as such it is essential that the threats facing a system are documented and prioritized so that testing can be made to address each risk area.

Where the previous levels of security testing addressed elements that could be tested before development had begun, the reviewing of source code is an element of testing that is performed during and after the development of code. This level of testing refers to the reviewing of code implementations to identify errors and vulnerabilities that must be addressed in the written code (OWASP, 2020). This could include testing as code is being written as well as in larger functional components to identify coding vulnerabilities (Merkow & Raghavan, 2012). These vulnerabilities could include deprecated code, coding mistakes, or bad coding practices that many times would not be detected by automated testing processes, highlighting the importance of combined automated and manual code reviews (OWASP, 2020). Testing at a source code level provides an important view of functionality for security professionals as the source code provides an in-depth representation of how functionality is implemented allowing for the better detection of where vulnerabilities may exist (OWASP, 2020). However, source

code reviews are only representative of white box testing where complete knowledge of the system is provided in testing (Tsui, Karam, Bernal, 2016). For this reason, it may be beneficial to additionally perform black box testing in which security professionals take on an adversarial mindset and attempt to hack a system without knowledge of the internal code structure. Source code is the level of abstraction on which development occurs, and for this reason, it is essential for security professionals to perform reviews on this level and ensure that security objectives are fulfilled within code implementations (OWASP, 2020).

The final development level, penetration testing, then represents the black box testing approach where adversarial thinking is applied to a finished project to test a hacker's ability to surpass a project's security features (OWASP, 2020). In this way, an understanding of the effectiveness of security implementations can be achieved and decisions can be made as to areas that require refactoring or improvement (Merkow & Raghavan, 2012). This form of testing additionally carries the added benefit of allowing external parties to potentially highlight areas of concern that had not been previously addressed in the security implementations (OWASP, 2020). Penetration testing can be limited in the scope of vulnerabilities that it can address, but in certain projects such as web applications, can be a valuable tool to identify the way a system responds to a threat in action. Each of the levels of security testing approaches different aspects of system security and through the application of testing at each level, a comprehensive review of security effectiveness can be performed.

Develop a Stride Threat Model:

The STRIDE threat modeling format is a way for a security professional to model the threats facing a system and provides a framework from which testing can be performed (Microsoft, 2019). The letters of the acronym STRIDE stand for each of the core focus areas of the threat assessment being spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privileges (NCSC, 2022). Understanding and modeling threats relevant to these domains can then provide a development team with a shared understanding of risks facing a system and allow for the creation of test cases to harden system security to these threats. For this reason, the STRIDE threat model has been implemented to portray the threats facing the Hackazon application which will then be used as the basis of test case formation.

The first area of the STRIDE threat model, spoofing, refers to a malicious actor's ability to access and then utilize the authentication credentials of a user other than their own (Microsoft, 2019). Broken access control is a major threat facing the Hackazon application, and as such system areas regarding the authentication of users, where authentication occurs, data storage, and data transportation must be examined, and test cases for each of these areas formed (NCSC, 2022). Spoofing can then be seen to be tested for within the Hackazon application through the examination of authentication functions throughout the application. These areas should properly conceal user credentials when stored or transported (NCSC, 2022) and additionally malicious actors should be stopped from submitting false credentials. From this, test cases can be developed to perform penetration testing on authentication points and ensure that access control cannot be broken by spoofing. Spoofing is one of the primary threat areas facing the Hackazon application, and as such resources should be dedicated to this threat vector to provide sufficient testing of all access points.

The next threat domain, tampering, can then be seen to represent a malicious actor's ability to modify system data (Microsoft, 2019). Within the Hackazon application, user data could represent personal information but also data values such as user privileges or session IDs. Tampering can then be seen to have severe implications as the modification of user privileges or session IDs could result in session hijacking or broken access control (NCSC, 2022). From this, test cases can then be developed to test against the modification of data in source code as well as in the application. This could include the testing of hash value authentication where hash values are used as well as the enforcement of thorough access control whenever data value modifications are attempted (NCSC, 2022).

Following this, the next threat area of repudiation can be examined which addresses the ability for system users to deny having taken part in an action (Microsoft, 2019). This threat can mostly be mitigated through strong implementations of both policy and technical implementations and as such, test cases can be generated to examine both of these implementation domains. Within the Hackazon context, this could be observed as ensuring proper encryption technologies to provide nonrepudiation are implemented such as the enforcement of digital signatures and certificates (Basta, 2018). Further, accurate and protected system logs can greatly contribute to nonrepudiation within a system (Merkow & Raghavan, 2012) and as such, policy regarding the implementation of logs can be examined to ensure that a sufficient amount of logging is taking place. This can then further be tested in the application itself to ensure that logs are accurately taken, and signature validation is performed when a user initiates an action request.

This then leads to the next threat area to be modeled being the area of information disclosure. Information disclosure refers to a violation of data confidentiality in which data is

exposed to a user who is not its owner (Microsoft, 2019). This threat can occur due to errors in implementation, data storage, or malicious action and as such should be tested through all areas of implementation. Within the Hackazon application, this then can be seen as a thorough examination of source code including the areas of code that send database requests to ensure that user validation is enforced at each of these areas. Additionally, penetration testing can be performed to ensure that an incorrect presentation of user data does not occur throughout the system. Information disclosure is a serious threat facing the Hackazon application as sensitive user data such as payment and personal details are stored and the fact that a breach of this data could open the organization up to litigation. For this reason, a large portion of testing resources should be committed to the sphere of testing against improper information disclosures.

Another threat domain that must be addressed and modeled for test cases within the Hackazon application is exposure to denial-of-service (DoS) attacks (Microsoft, 2019). As Hackazon is a web-based application, the loss of services could have serious implications on costs and reputation highlighting this threat area as a major area of system concern. For this reason, test cases must be generated to discover how vulnerable the system is to DoS attacks and determine if additional mitigation strategies are needed (NCSC, 2022). With this in mind, it can be seen that the primary testing domain for this threat area is penetration testing as the threat faces the application service itself. However, source code evaluation and policy examination for security response plans can ensure that sufficient mitigation exists within the organizational structure to defend against DoS attacks.

The final area of concern to be modeled through the STRIDE threat analysis can then be seen as the area of elevating privileges. The elevation of privileges is one of the primary vectors of attack as it can allow for powerful administrative actions to be taken (Merkow & Raghavan,

2012) highlighting the importance of sufficient testing to be performed so that privilege escalation cannot occur. Within the Hackazon context, each user group and privilege category should be thoroughly tested by creating accounts of varying privilege levels and ensuring access control cannot be broken with these groups. This includes extra controls being placed on administrative accounts as well as proper validation of administrative users and actions (NCSC, 2022). From this, the importance of testing all privilege escalation commands thoroughly can be seen and source code and penetration testing should be performed to ensure that privilege escalation can only occur within the approved context.

The threat model generated from the STRIDE framework can then serve as an artifact for the development team in generating specific test case implementations within the Hackazon context. Mitigating threats in these areas will vastly increase the security of the system highlighting the importance of conducting testing according to a shared testing framework and understanding (Tsui, Karam, Bernal, 2016). The application of the STRIDE threat model will then further allow for the measuring of testing metrics aiding the security professional and development team in determining the effectiveness of an implementation as well as the primary threats the system remains vulnerable to. These metrics are an essential part of the development process and can serve as guidance toward new implementations or mitigation efforts in further refactoring increasing the resilience of the deliverable software product (Merkow & Raghavan, 2012). This highlights the importance of modeling system threats and within the context of the Hackazon application will contribute to an improved security implementation.

Select Appropriate Security Test Cases:

SR ID	SR Name	STC ID	STC Name	STC Description, Constraints, and Comments
SR-IDEN-001	Unique User ID	STC-IDEN-001	Unique User ID	To better enforce access control, each user will be forced to create a unique user ID (Merkow & Raghavan, 2012). This requirement can be tested by attempting to create users with the same user ID as well as modifying a user ID to an existing ID. In both of these cases, the system should reject the process and inform the user to select a different user ID.
SR-IDEN-002	Backdoor Prevention	STC-IDEN-002-1	Backdoor Prevention 1	This test case is designed to verify that a user is logged in before being given access to system resources (Merkow & Raghavan, 2012). This use case can be tested by attempting to bypass the user authentication stage and ensuring that no attempt succeeds. This testing may then include penetration testing as attempts to access the system through bypassing verification will be made remotely by external parties.

SR-IDEN-002	Backdoor Prevention	STC-IDEN-002-2	Backdoor Prevention 2	This test case then reviews the source-level code to identify any logical errors through which access control could be broken (Merkow & Raghavan, 2012). This test must then examine all code that addresses access control features of interfaces and verify that access control cannot be broken from a theoretical standpoint.
SR-IDEN-003	Process Identifier Code/Accountability	STC-IDEN-003	Process Identifier Code/Accountability	This test case involves ensuring that all system processes are associated with a user ID to verify which user initiated the command as well as if the appropriate privileges are held to execute that command (Merkow & Raghavan, 2012). This test case can be performed by creating users of each user group and then attempting processes in each user group to ensure that process IDs are generated and that the proper access control is implemented.
SR-IDEN-004	Auto disable User IDs	STC-IDEN-004	Auto disable User IDs	The auto-disable test case will then seek to verify that a user ID will be disabled after a certain period of inactivity (Merkow & Raghavan, 2012). This test case can be performed by activating a user ID and then allowing that user ID to idle for the appropriate amount of time

				while observing the state of that ID. This can then allow for an understanding if the disabling feature is working correctly to be established.
SR-IDEN-005	Security Attributes	STC-IDEN-005	Security Attributes	The testing for security attribute compliance then refers to the correct assigning of user privileges to each user group (Merkow & Raghavan, 2012). This security feature can be tested by creating a user of each user group type and testing privileges both allowed and disallowed for that user to ensure correct access control principles are followed. Further, the review of source code regarding access control can allow for further examination of user group privileges allowing for confirmation that proper access control is implemented.
SR-ATEN-001	Credential Security	STC-ATEN-001	Credential Security	This security test case refers to the testing and verification that one-way hashes are applied to all passwords stored in the database (Merkow & Raghavan, 2012). This test case can be performed by creating a password for a user and examining the database entry to ensure that the appropriate hashing has occurred. Further, running the

				same password into the database with a second user can then confirm that appropriate salt is added to the hashes to ensure that identical passwords hash to different values for unique users.
SR-ATEN-002	Replay Attack Protection	STC-ATEN-002	Replay Attack Protection	Replay attack protection is an essential aspect of security for the Hackazon application and test cases can be generated to ensure that these attacks are not possible. These attacks can be tested for by attempting to deploy a replay attack on the system through penetration testing and observing if the attack can be performed. Further, analysis of the code sections that mitigate replay attacks can be investigated to verify that proper security practices are followed.
SR-ATEN-003	Protect Credential Guessing	STC-ATEN-003	Protect Credential Guessing	This test case refers to ensuring that error messages do not give too much information regarding the underlying implementation (Merkow & Raghavan, 2012). To test this, a variety of errors can be intentionally performed on the system and the error messages observed. If the system stores all of its error messages in a centralized file location, the examination of error codes and their

				associated triggers can also be performed to validate that error codes are not overly specific as written. Further, brute force attacks should be protected against (Merkow & Raghavan, 2012) and this can be tested by attempting to brute force a user's credentials and observing the system's response to this attack.
SR-ATEN-005	Reauthentication	STC-ATEN-005	Reauthentication	For this test case, a reauthentication of user credentials must be performed at each critical section of the application (Merkow & Raghavan, 2012). This test can be performed by attempting to access each critical section of the application and ensuring that proper reauthentication is enforced.
SR-ATEN-006	Protection of Credentials	STC-ATEN-006	Protection of Credentials	To verify that the user's credentials are not sent in plain text, the test case can be performed where a penetration tester sniffs network traffic while submitting user credentials through each relevant point in the application. This observation will then allow for an understanding if encryption is being performed at each critical point as well as if the appropriate transfer protocols are being used.

SR-ATEN-007	Password Changes	STC-ATEN-007	Password Changes	For this test case, specific criteria regarding password change policy must be verified as enforced within the Hackazon application (Merkow & Raghavan, 2012). These criteria include the change of a user's password upon first login and at will later on. Further, the forced avoidance of password change must not be possible, and all user interaction must lead to the password change page before further user action can be taken. This can be tested by creating a user account relevant to each stage of password change and ensuring the correct procedures are enforced.
SR-ATEN-008	Password Aging	STC-ATEN-008	Password Aging	All user passwords should expire after a given amount of time and this must be tested both logically and in practice within the system (Merkow & Raghavan, 2012). To test this, a user account can be created, and the password for that account aged within the database to observe if the system enforces password modification. Further testing can then be performed within the context of actual time to ensure expiration occurs as intended.

SR-ATEN-010	Secure Password Changes	STC-ATEN-010	Secure Password Changes	When a user attempts to change their password, a reauthentication of the user's credentials should be performed (Merkow & Raghavan, 2012). This test case can then be easily performed by creating a user and attempting to change the password for that user.
SR-ATEN-013	Configurable False Accept/Rejection	STC-ATEN-013	Configurable False Accept/Rejection	An automated testing approach can be performed to test for false acceptance and rejection cases and an evaluation of these cases could then be performed. Ideally, no false acceptance or rejections should be performed, but given the choice the system should be seen to prioritize rejection and this feature can also be validated through this testing.
SR-AUTR-002	Access Rights	STC-AUTR-001-1	Access Rights 1	This test case involves the system validating that a user holds the appropriate access rights before being granted access to resources. To test this, user accounts of varying access rights can be created and attempt to access resources they both do and do not have access rights for. An evaluation of the effectiveness of the control can then be performed as access control should not be broken.

SR-AUTR-002	Access Rights	STC-AUTR-001-2	Access Rights 2	Differing from the previous test case, this test case involves the ability for a malicious actor to brute force web page URLs until access is gained to a restricted part of the application (Merkow & Raghavan, 2012). This can be tested for by taking the map of the application's URLs and attempting to break access control by manually entering those URLs into the browser. It can then be confirmed if the proper controls are in place to mitigate this risk.
SR-AUTR-002	Access Rights	STC-AUTR-001-3	Access Rights 3	Additionally, the system should not allow access for users to hidden parameters (Merkow & Raghavan, 2012) and this can be tested for by examining the source code to identify any parameters that may exist within the application and then trying to access them through penetration testing.
SR-AUTR-003	Account Lockout	STC-AUTR-003	Account Lockout	When consecutive incorrect login attempts are made, the system should lock the user out until further verification can be performed (Merkow & Raghavan, 2012). This can be tested by attempting to log in to a user account with incorrect credentials and observing the

				outcome. This test would then be marked as successful if the lockout occurs on the correct login attempt and the user is informed that they must further verify themselves.
SR-AUTR-006	Session Timeout	STC-AUTR-006	Session Timeout	This test will then ensure that a timeout feature is present and functioning within the application (Merkow & Raghavan, 2012). When a user has been inactive for a given period of time, they should be signed out of the system, and this can be tested through creating a user account and then observing when the timeout feature kicks in for that user. The time to logout is a measurable component of this test case and can be compared against the intended time to ensure compliance.
SR-AUTR-008	User and Group Privileges	STC-AUTR-008	User and Group Privileges	The system must maintain the capability to implement user access groups (Merkow & Raghavan, 2012) and this can be verified by inspecting the source code as well as the administrative interfaces for implementing such controls.

SR-AUTR-009	Role-Based Access Control (RBAC)	STC-AUTR-009	Role-Based Access Control (RBAC)	Like the test case above, this test case involves further exploration to ensure that appropriate user classes are created and that the appropriate privileges are assigned to each user class. These user classes can be compared against the requirements documentation to ensure that the principle of least privilege is being followed in implementation. This can be further verified through interviews to ensure features are not granted to user groups that do not need them.
SR-AUDT-001	Audit Log	STC-AUDT-001	Audit Log	The system should maintain an audit log that logs all relevant security events (Merkow & Raghavan, 2012). This can be tested by first confirming that the audit log exists, and then committing security-related events to ensure the events are properly logged.
SR-AUDT-003	Logging of Specific Events	STC-AUDT-003	Logging of Specific Events	As an extension of the previous test case, this test case furthers the exploration by looking at specific cases that need to be logged. These specific cases can be performed in various ways to ensure that logging occurs in every circumstance needed.

SR-AUDT-005	Archival of Audit Logs	STC-AUDT-005	Archival of Audit Logs	Similarly, the archival of logs must be performed with logs being stored for the appropriate amount of time (Merkow & Raghavan, 2012). This feature can be tested through the creation and aging of log entries to ensure the proper procedures are followed. Further, source code evaluation can additionally confirm that these procedures are implemented as intended.
SR-AUDT-008	Protection of Audit Log	STC-AUDT-008	Protection of Audit Log	As audit logs are a prime target of malicious actors, audit logs must not be modifiable by users including administrators (Merkow & Raghavan, 2012). An internal penetration test can be performed to attempt to modify these logs and a verification of the outcome can then be observed.
SR-INTG-001	Integrity Checking	STC-INTG-001	Integrity Checking	This test case references the ability of the system to perform integrity checking in any area where data is transmitted and received (Merkow & Raghavan, 2012). In these areas, a thorough evaluation of the integrity-checking process can be performed and from this analysis, an understanding of compliance can be gained. These integrity

				checks could also be performed at the source code level to ensure that proper enforcement is implemented in the code.
SR-INTG-002	Source Identification	STC-INTG-002	Source Identification	Any time an address is passed to the application, the application must be able to verify the source of that destination (Merkow & Raghavan, 2012). This ability can be tested by attempting to perform a server-side request forgery attack and documenting the results. The system should not allow these attacks to occur and the proper rejection actions should be taken.
SR-INTG-005	Integrity of Sensitive Information	STC-INTG-005	Integrity of Sensitive Information	For this test case, a thorough examination of the transport protocols used in transporting information can be performed. This can be tested at all points where sensitive information is transported and in all cases the data should be transported by secure transfer protocols with no known vulnerabilities. Any non-compliant protocols should then be brought into compliance and changed to validated protocols.
SR-INTG-007	Integrity Checks	STC-INTG-007	Integrity Checks	The Hackazon application must have the ability to perform integrity checks and this feature can be tested by observing all receiving

				points for data and verifying that integrity checks are enforced at each. This will primarily be observed through source code examination but can also be observed in attempting to transfer code that has been modified en route and observing the system's reaction to such code.

References

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

DSIT. (2023). *Conducting a STRIDE based threat analysis*. Department for Science, Innovation, and Technology. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1155778/Conducting_a_STRIDE-based_threat_analysis.pdf

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

Microsoft. (August 25, 2022). *Microsoft threat modeling tool: Threats*. Microsoft. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats

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

OWASP. (March 12, 2020). *OWASP: Web security testing guide*. OWASP. https://owasp.org/www-project-web-security-testing-guide/

Salva, S., & Regainia, L. (2019). An approach for guiding developers in the choice of security solutions and in the generation of concrete test cases. *Software Quality Journal, 27*(2), 675-701. https://doi.org/10.1007/s11219-018-9438-2

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

Wang, C., Pastore, F., Goknil, A., & Briand, L. C. (2022). Automatic generation of acceptance test cases from use case specifications: An NLP-based approach. *IEEE Transactions on Software Engineering, 48*(2), 585-616. https://doi.org/10.1109/TSE.2020.2998503