Article

Not peer-reviewed version

Maximizing test coverage for security threats Using Optimal Test Data Generation

Talha Hussain , Rizwan Bin Faiz , Mohammad Aljaidi * , Adnan Khattak , Ghassan Samara , Ayoub Alsarhan , Raed Alazaidah

Posted Date: 5 May 2023

doi: 10.20944/preprints202305.0343.v1

Keywords: Modified Condition/Decision Coverage; Decision Coverage; Test Coverage; Test Data; Object Constraint Language; Structured Misuse Case Description.

Article

Maximizing Test Coverage for Security Threats Using Optimal Test Data Generation

Talha Hussain [1], Rizwan Bin Faiz [1], Mohammad Aljaidi [2,*], Adnan Khattak [1], Ghassan Samara [2], Ayoub Alsarhan [3], Raed Alazaidah [2]

[1] Faculty of Computing, Riphah International University, Islamabad, Pakistan, Emails: thussain98ml@gmail.com (T.H.), rizwan.faiz@riphah.edu.pk (R. B. F.), adnan_ktk08@yahoo.com (A.K.)
[2] Department of Computer Science, Faculty of Information Technology, Zarqa University, Zarqa 13110 Jordan, Emails: mjaidi@zu.edu.jo (M.A.), gsamara@zu.edu.jo (G.S.), raed.azaedah@zu.edu.jo (R.A.)
[3] Department of Information Technology, Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology, The Hashemite University, Zarqa 13116, Jordan, ayoubm@hu.edu.jo (A.A.)
* Correspondence should be addressed to mjaidi@zu.edu.jo (M.A.)

Abstract: As time continues to advance, the need for robust security threat mitigation has become increasingly vital in software. However, ensuring early effective security threat mitigation requires optimal test data and consistent test case design. It is a constant struggle to maximize test coverage through test data optimization. We conducted explanatory research to maximize test coverage of security requirements as modeled in Structured Misuse Case Description (SMCD) i.e., structured specification of misuse case, so as to improve consistency in optimal test data generation. We specified constraints upon Mal activity in Object Constraint Language (OCL) in order to minimize human dependency and improve *consistency* in optimal test data generation. It was evident through results that MC/DC generated optimal test data of security threats through SMCD in comparison to the Decision Coverage method thus resulting in designing a significantly lower number of test cases and yet maximizing test coverage of security threats. MC/DC generated test data with n+1, while Decision Coverage generated test data with $\mathbf{2^n}$, we, therefore, conclude that MC/DC maximizes test coverage through optimal test data from SMCD in comparison to Decision Coverage.

Keywords: Modified Condition/Decision Coverage; Decision Coverage; test coverage; test data; Object Constraint Language; Structured Misuse Case Description

1. Introduction

Software testing is a time-consuming but vital activity that tries to raise the quality of software[1].The primary goal of testing is to ensure that the delivered product is bug-free[2]. Software testing is crucial during the development process since it seeks to show that the program is free of errors[3]. Test coverage is a crucial component of software maintenance and a significant indicator of software testing[4]. Offering data on various coverage items aids in evaluating the success of the testing process. The hardest part of maximizing test coverage is coming up with appropriate test data[5]. Moreover, generating test data to achieve 100% coverage is labor-intensive and expensive[6]. A greater number of tests also necessitates a longer test period and greater tester memory. Test development becomes more challenging due to the growing number of tests required to fulfil adequate coverage criteria. Selecting a portion of tests from a large baseline test set becomes critical when it is impossible to apply all of the (supposedly Offering required) tests due to test time or tester memory limitations[7]. It is clear from the literature that testing accounts for more than 40% of project costs[8]. Instead of covering all the entities in the tested program, test data can be aimed toward covering the optimal set to reduce the testing effort[9]. Reducing the amount of data utilized for testing is one method of minimizing this cost. Test data optimization is a method for minimizing test data sets that can be used for both black-box and white-box testing[10]. Our research goal is to maximize test coverage through optimal test data generation at the design level for security threat mitigation.

We tested the UML model having constraints specified in Object Constraint Language (OCL) as an experiment to use our approach to get the best test data and test coverage because we have evidence from the literature that UML models with specified constraints in OCL are comprehensive and consistent [11]. The Modified Condition/Decision Coverage MC/DC methodology is one widely used method for generating the optimum test data for code coverage [12]. Through optimal test data generation, we aim to maximize test coverage in our study. Achieving 100% test coverage can be thought of as maximizing test coverage. Since it is noted in the literature that the method used to generate test data to reach 100% coverage is laborious and expensive, achieving 100% test coverage is not practicable.

Use cases outline functional specifications; however, they are unable to allow the modeling of security threats. In order to model security threats and identify misuse cases, the use case diagram is developed. Since a use case model simply specifies the necessary capabilities, the textual description often captures the core of the use case. This textual description plays a vital role while representing a misuse case [13]. Diagrams of misuse cases and the textual descriptions of those diagrams give developer essential security-related details.

To deal with malicious system usage, we need security standards. Similar to functional requirements, security threats must be addressed. The system's security can be enhanced by adopting security requirements [14]. Misuse cases can be used to implement security threats, and test cases are developed from misuse cases for security threats. Diagrams of misuse cases and the textual descriptions of those diagrams give the developer essential security-related details. Techniques to create security test cases from misuse instances have been proposed by a plethora of authors. By executing test cases and verifying that the software worked as intended, security testing may be put into practice utilizing misuse cases. An example of a test case would include test input, excepted output, and actual output [15]. It is a sign that the program functionality is correctly implemented when the expected output and actual output match.

The significance of our research is that we have designed consistent acceptance test cases for security threats (authentication and authorization) through structured misuse case descriptions for early-stage mitigation of security threats. This will help us to overcome the challenge of inconsisent acceptance test case design due to its reliance upon human judgment. For that, we have first identified misusage scenarios through structured misuse case descriptions against security threats such as authentication and authorization and their corresponding mitigation. Then to identify the inputs and triggers of each misusage scenario of security threats, we have modelled them in Mal-activity diagrams. In the third step, we have designed the acceptance test cases for security threats, i.e., authentication and authorization through SMCD. The result shows that test cases designed through MC/DC are optimal and require minimum test conditions.

We create test cases using (i) automated and brute, (ii) weak password attacks, (iii) session id links, and (iv) session expiry time. The goal of employing misuse situations is to build security acceptance test cases at a high level and to get beyond the challenge of discovering acceptance test cases from programming languages. Without any knowledge of the programming language, the user will concentrate on creating acceptance test cases from textual descriptions. To create acceptance test cases, textual descriptions of the use cases, misuse instances, threats to them, and solutions are identified. The various usage scenarios should be covered by the designed acceptance test cases. Security acceptance test cases include expected outputs and inputs like any other test case. Data or functional calls might be used in the input security acceptance test scenarios. Evaluation of acceptance test cases is done in the output of the intended outcome.

The remaining part of section of this document is structured as follows: Literature Review, Research Question and Gap analysis is discussed in Section 2 and Research Methodology is discussed in Section 3. Experiment design along with entire implementation process for identifying misuse case scenarios, associating threats using SMCD, designing malicious activity, producing test data using MC/DC and D/C, capturing the inputs and triggers used in scenarios, and designing acceptance test cases for the misuse case scenarios are mentioned in Section 4. The acceptance test cases created for security risks from both SMCD and USMCD are also evaluated consistently in Section 4 of the report. Results, threats to validity and future work are discussed in Section 5. Conclusion and future work mentioned in last Section 6.

 doi:10.20944/preprints202305.0343.v1

2. Literature Review

The literature has covered test case prioritization extensively. Amita Jain presented a system that prioritizes the independent pathways used during the path testing process. Test Coverage is a crucial component of software maintenance and a significant indicator of the quality of the product. By offering information on various coverage topics, it aids in evaluating the success of the testing process. It takes a lot of time and money to create enough test data to reach 100% coverage. A greater number of tests necessitates a longer testing period and more tester memory. However, the quantity of tests needed to guarantee high coverage criteria has been rising, making test development more difficult[16]. A subset of tests from a large baseline test set must be chosen when it is difficult to apply all the (alleged) tests due to test time or tester memory limitations. Test data generation can be aimed towards covering the ideal set rather of all the entities in the tested program in order to reduce the testing effort. We generate a large amount of test data when we conduct model-based testing. The main challenge in software testing is coming up with test data that meets a specific adequacy criterion. As we will have different test data at the design level, more precisely when we conduct the testing manually, this problem results from the test case subjectivity issue. The test case subjectivity issue will be fixed because the design diagram was made precise with constraints specified in OCL language [11]. Techniques like MC/DC and Decision coverage are used at the code level to increase test coverage. It hasn't yet been determined in the literature review which of these two maximizes test coverage through ideal test data creation at the design level, so we'll use both of these strategies and compare the outcomes.

In [17], the author encourages us to use MC/DC at the design level since it increases test coverage by creating test data at the code level. In [18] author, independently applied the suggested methodology of producing test data using the MC/DC strategy for a search-based empirical evaluation. The literature makes a strong case for the value of constraints specified in OCL at the design level. By outlining models that cannot be included, Constraints specified in OCL help UML models be comprehensive, consistent, and accurate when used with them [11]. The unified modelling language (UML), which receives a lot of interest from scholars and professionals, has become a crucial standard for modelling software systems [19]. Extensive test data is produced when we conduct model-based testing[20]. Making test data that meet a specified adequacy criterion is a key challenge in software testing.

In [21] author has generated acceptance test cases for security threats using an unstructured misuse case description. By incorporating the misuse case description into the mal-activity diagram, a misuse scenario is created and inputs and triggers are determined. The flow of usage shown in the mal-activity diagram is used to design test cases. Test cases are created in [22] where misuse instances pose security issues. Students that attend the university participate in the experiments. On a web application for course management, the experiment is run. The findings show that the suggested misuse case creation method offers superior coverage for security issues. Their strategy, nevertheless, has to be better organized and provide a thorough explanation of the procedure. The techniques to generate test cases from the use case were proposed in [23]. Web-based apps were validated with the assistance of the provided method. A use case model is initially created from the functional requirements. As a result of the use case model, test cases are created. The final stage involves using commercially available tools to execute the prepared test cases. In [24] a useful method for generating test sequences based on models was described. Using threat models, this method produces extensive threat trees. By taking into account both valid and incorrect test input, executable test cases and the security test sequence are built from the threat tree. In [25], author proposed an approach to generate test cases from use cases, misuse cases, and 137 mitigation of use case descriptions. Early on in a product's development, this includes security features. In misuse scenarios, they recommend several improvements to make it easier to define security needs.

2.1. Research Question

The main challenge in software testing is coming up with test data that meets an established adequacy criterion. It is evident from the literature, that test coverage in source code can be

 doi:10.20944/preprints202305.0343.v1

maximized through optimal test data generation using MC/DC and Decision Coverage Approach [17]. However, there is no evidence in the literature on how to maximize test coverage for SMCD through optimal test data. To achieve our goal, we focus on the following research questions:

- RQ1**:** Which among Decision Coverage and MC/DC maximizes test coverage for security threats in Structured Misuse case Description?
- Hypothesis 1: Decision Coverage maximizes test coverage for security threats in Structured Misuse case Description
- Hypothesis 2: MC/DC maximizes test coverage for security threats in Structured Misuse case Description

3. Research Methodology

This section defines the thorough facts of our experiment, which we carry out. Let us discuss the content in detail. Our research approach is quantitative because we have a numerical dataset and we will use statistical analysis methods to test relationships between variables. Applying MC/DC and Decision coverage criteria on Design Level for optimal test coverage of security requirements generating Optimal Test and improving consistency for security requirements. We will perform an experiment to generate optimal test coverage to reduce the testing effort by achieving maximum test coverage through optimal test data.

Our research methodology is explanatory research, which aims to explain the causes and consequences of a well-defined problem. Spending too much time in testing is a well-defined problem; we will check the consistency of acceptance test cases for structured and unstructured misuse case in first experiment. Similarly, for second experiment optimal test data generation using MC/DC and Decision coverage approach for optimal test data generation.

There are often independent and dependent variables in a control experiment. Our research uses the experimental method to determine the cause and effect between variables, including dependent and independent variables. In our experiment, the dependent variable is the Optimal Set of test data, and the independent variables are MC/DC and Decision Coverage.

4. Experiment Design

This study aims to Maximize test coverage for security threats using Optimal Test Data Generation. One of the most common way to model security threats is through misuse case. In order to maximize test coverage at the design level we generated test data through Modified Condition Decision Coverage and Decision Coverage. In order to achieve our research goal, we undertake an experiment to generate optimal test data. In the experiment, we generated test data through Modified Condition Decision Coverage (MC/DC) and Decision Coverage from misuse case diagrams, with test conditions extracted for seven constraints. Overall, this study contributes to the field of software security by providing a methodology for generating optimal test data at the design level for security threats.

Steps of proposed methodology are as follows:

1. Identify Security authentication and authorization threats.
2. Design structured Misuse case Description.
3. Draw Mal Activity from Structured Misuse case Description.
4. Specify constraints in Mal Activity Diagram using OCL.
5. Transform Constraints into Boolean Expression.
6. Transform Boolean expression into Truth Table Expression.
7. Generate possible test data.
   7.1 Through MC/DC (Modified Condition Decision Coverage)
   7.2 Through Decision Coverage (D/C)
8 Compare and Find Optimal test data generated through MC/DC and D/C

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

9 Design Test Cases for generated optimal test data.

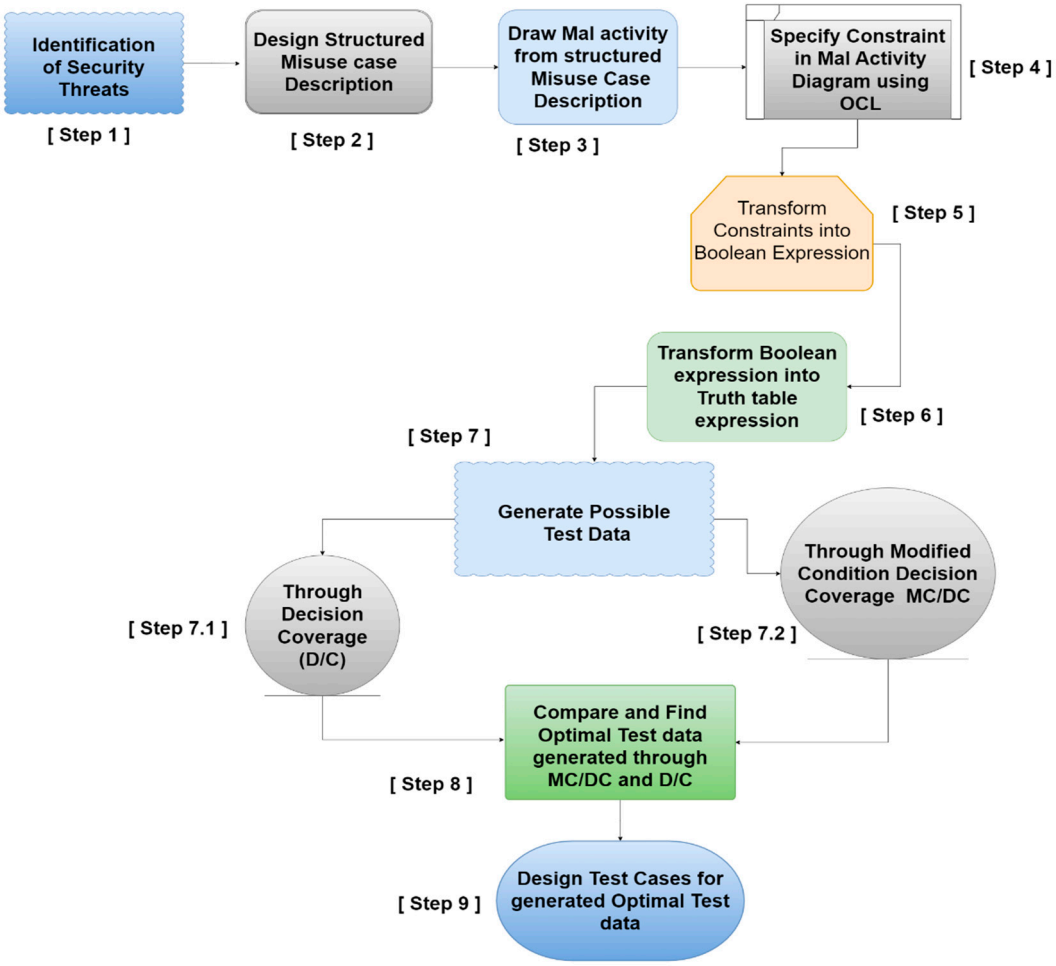

Figure 1. Proposed Methodology.

Step 1: Elicitation/Extraction of Security Requirements i.e. authentication and authorization

In the context of software development, security requirements refer to the set of features and functionalities that ensure that the software system is secure and protected against unauthorized access, data breaches, and other types of cyber-attacks. The first step in developing a secure software system is to elicit or extract the security requirements.

"Elicitation/Extraction of Security Requirements" refers to the process of identifying and defining these security requirements. In Table 1 Authentication and Authorization were addressed.

Table 1. Identification of Security Threats.

Elicitation of Security Requirement	Goal	Sub-Goal
The application should authenticate the User using a valid username and Password.	Security	Authentication
Authorization codes should be set up and modified only by the System Administrator.	Security	Authorization

Step 02: Structured Misuse case Description

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

Structured misuse case description is provided in following Table 2.

Table 2. Structured Misuse Case Description.

ID*	SMC-SA-001
Goal*	Security
Sub Goal*	Authentication
Misuse case Name*	Steal Login Details IMPLEMENTS steal sensitive data
Associated Misusers*	Information Thief
Author Name	ABC
Date	dd/mm/yy
Description*	Misuser gets access through automated attacks such as credential stuffing and brute force technique to Login into the system to perform illegal activities with the user data.
Preconditions*	The login page is accessible to the Information Thief.
Trigger*	Information Thief clicks the login button
Basic Flow*	Information Thief uses an automated attack tool to generate many combinations of usernames and passwords. Login Details, i.e., username, Password, and Captcha matched with the login details of the system. On Successful Login, a verification code will be sent on the user email id/SMS for multifactor authentication. If a user receives a verification code and verifies the login attempt, then the Information thief will be redirected to the User's personal and sensitive data pages.
Alternate Flow*	**BF-2**. In case of non-authentic/invalid login details, i.e., username, Password, and Captcha, or the number of login attempts are greater than three against the same IP, it will be blocked. **BF-4**. If the multifactor authentication verification code is not received through Email/SMS, repeat the Bf3.
Assumption	The system has login forms feeding input into database queries.
Threatens Use case*	User Login
Business Rules	The Hospital system shall be available to its end-users over the internet.
Stakeholder & Threats	Hospital O/I Maintenance Department, O/I User Department, Store Keeper, Dispenser. If deleted, data loss reveals sensitive information to damage the business and reputation of the hospital.
Threatens Use case Mitigation	If a user from the same IP address attempt three logins failed attempts, block the IP. Also, apply

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

	Captcha and multifactor authentication to avoid attacks.

Step 03 Designing Mal-Activity Diagram

In this step Mal-activity diagram is designed for Structured Misuse case Description. Structured Misuse case descriptions are modelled into the mal-Activity diagram to identify the inputs and triggers. Figure 2 illustrates the mal activity diagram modelled from the structured misuse description.

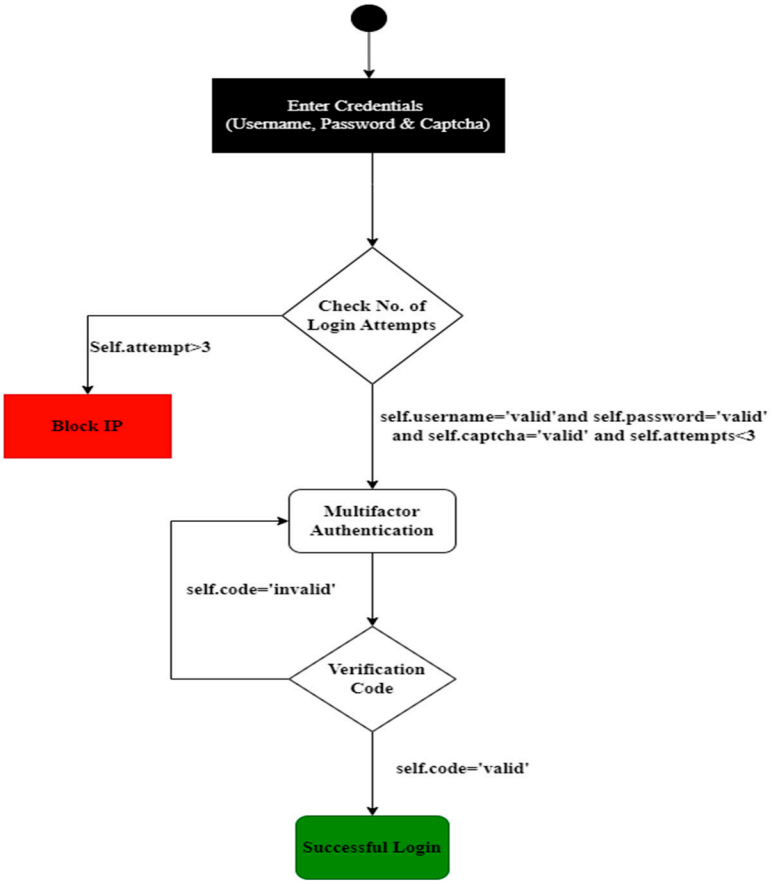

Figure 2. Mal-activity diagram for SMC-SA-001.

Step 4 Specify constraints in Mal Activity Diagram using OCL

Following is specification of constraint in OCL that will be used for transformation into boolean expression.

self. Username = 'valid' and self. Password='valid' and self.captcha='valid'} and self.attempts<3

Step 5 Transformation of Constraints into Boolean Expression

self. Username = 'valid' ∧ self. Password='valid') ∧ (self.captcha='valid' ∧ self.attempts<3))

After converting into logical operation, we applied the MC/DC and Decision Coverage to each constraint.

Step 6 Transform Boolean expression into truth table expression

Multiple solutions are required corresponding to a constraint to generate test data according to the MC/DC and D/C criterion. For example, consider a constraint as an expression C = A ∨ B, where A and B are the clauses of the constraint. There are four combinations of possible outcomes, two each for p and q (TT, TF, FF, and FT). To reformulate an constraint for MC/DC and Decision Coverage,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

we first identify the truth-value combinations required for MC/DC and Decision coverage of a given constraint. For this purpose, we use the pair-table approach suggested by Chilenski and Miller[26].

To solve the constraint, we will specify them as following. In our experiment constraints are specified as following in Table 3

Table 3. Explanation of constraints transformation.

a	self. Username = 'valid'
b	self. Password='valid'
c	self.captcha='valid'
d	self.attempts<3

Constraints specified using Table 3 becomes **((a ∧ b) ∧ (c ∧ d))**.

In following Table 4, the Boolean expression, which is first transformed from constraint, is now transformed into truth Table form.

Table 4. Transformation of Boolean Expression into truth table form.

N0.	a	b	c	d	((a ∧ b) ∧ (c ∧ d))
1	F	F	F	F	F
2	F	F	F	T	F
3	F	F	T	F	F
4	F	F	T	T	F
5	F	T	F	F	F
6	F	T	F	T	F
7	F	T	T	F	F
8	F	T	T	T	F
9	T	F	F	F	F
10	T	F	F	T	F
11	T	F	T	F	F
12	T	F	T	T	F
13	T	T	F	F	F
14	T	T	F	T	F
15	T	T	T	F	F
16	T	T	T	T	T

Step 7 Generating Possible Test

We will use the Decision Coverage approach mentioned in [27] and MC/DC described by Chilenski and Miller[26]. To obtain MC/DC and Decision Coverage, we need to solve the combinations of true and false values required to achieve the MC/DC criterion. Multiple solutions are required corresponding to a constraint to generate test data according to the MC/DC criterion. For example, consider a constraint as an expression C = p ∨ q, where p and q are the clauses of the constraint. There are four combinations of possible outcomes, two each for p and q (TT, TF, FF, and FT).

Step 7.1 Generating test data for MC/DC

The idea of the MC/DC is to select the subset of all possible combinations that directly impact the outcome value of the actual constraint. In the case of C, these combinations are (FF, TF, and FT).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

To identify this subset, the first step is to reformulate the original constraint to obtain more constraints that satisfy the MC/DC criterion.

The pair table suggested in [26] provides several potential pairs for each clause, and we need to select minimum subsets of pairs that cover all clauses. In our table potential pairs are (8,16) from A , (12,16) from B , (14,16) from C and, (15,26) from D. For A, we need to test only conditions of (8,16); for B, we need to test only (12,16). For C, we need to test only (14,16); for D, we must only test (15,16). So, we need to test 5 Constraints (8, 12, 14, 15, 16) as specified in Table 5.

Table 5. Color Mapping for potential Constraints found in MC/DC.

N0.	a	b	c	d	((a ∧ b) ∧ (c ∧ d))
1	F	F	F	F	F
2	F	F	F	T	F
3	F	F	T	F	F
4	F	F	T	T	F
5	F	T	F	F	F
6	F	T	F	T	F
7	F	T	T	F	F
8	F	T	T	T	F
9	T	F	F	F	F
10	T	F	F	T	F
11	T	F	T	F	F
12	T	F	T	T	F
13	T	T	F	F	F
14	T	T	F	T	F
15	T	T	T	F	F
16	T	T	T	T	T

Step 7.2 Generating test data for D/C

Decision coverage requires test cases to cover both branches of a decision. For each decision, the D/C criterion requires two test cases. For example, for a decision, Boolean expression requires two test cases, e.g., the test cases (A=true, A =False) [17].

We have 16 test combinations for decision coverage because we have to test whether each combination is false or true for each condition, as mentioned in [17].

Step 08 Compare and Find Optimal Test Data

In the above example, we have 16 test conditions; for Decision coverage, we need to test all 16 test conditions to achieve maximum coverage for MC/DC. Therefore, it is required to test only 5 test conditions which comprise the actual test conditions in terms of finding maximum errors. For MC/DC, optimal test conditions were found, and for Decision Coverage, it is required to test all conditions. So our hypothesis 2 'MC/DC maximizes test coverage for security threats in Structured Misuse case Description' is true and hypothesis 1 will be negated in the results of this experiment.

In step 08, we find that test data generated for MC/DC requires fewer test combinations as compared to Decision Coverage. Therefore In test case design we will use test combinations identified for MC/DC.

4.2. Test Combinations Identified by MC/DC:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

There were total 16 constraints generated truth table as mentioned in Table 4. After applying MC/DC found test conditions in MC/DC are 8,12,14,15,16 as specified in Table 5. These constraints are identified after application of MC/DC on relevant OCL value for A,B,C,D. In following Table 3 color mapping is given for relevant OCL value i.e for A,B,C,D. For **a** potential constraints are 8,16 mapped in orange color. Similarly, for **b** potential constraints are 12,16 mapped in grey colour, for **c** 14,16 mapped in green and for **d** for 15,16 in purple. In result we found that the test combinations are 8,12,14,15,16 as specified in Table 5, for which the test case will be designed.

Step 09 Acceptance test case design through Structured Misuse Case Description

In step 08 we identified that potentional constraints are 8,12,14,15 and 16. In order to design test case from these constraint we need to check the truth/false value for each a,b,c,d truth table value. In case if the value is true constraint will remain same, incase if it's false it will be reversed e.g in given 8 constraint we have f,t,t,t values for a,b,c and d respectively. We need to modify the constraint value for a since we got F result for potentional 8th constraint. In original Constrain we have a= self. Username = 'valid' referred to Table 1, after altering the F value it will become self. Username = 'Invalid' and rest of the constraint will remain same. For all these potential constraint we have updated the original constraint accordingly in following Table 6.

Table 6. Identified Test Combination for MC/DC.

Constraint No.	Test Scenario	Constraint
8	Invalid Username Unsuccessful login	self. Username = 'Invalid' ∧ self. Password='valid') ∧ (self.captcha='valid' ∧ self.attempts<3)
12	Unsuccessful Login due to invalid Password	self. Username = 'valid' ∧ self. Password='Invalid') ∧ (self.captcha='valid' ∧ self.attempts<3))
14	Unsuccessful Login due to invalid Captcha	self. Username = 'valid' ∧ self. Password='valid') ∧ (self.captcha='Invalid' ∧ self.attempts<3))
15	Unsuccessful Login due to more than three login attempts	self. Username = 'valid' ∧ self. Password='valid') ∧ (self.captcha='valid' ∧ self.attempts>3))
16	Successful Login	self. Username = 'valid' ∧ self. Password='valid') ∧ (self.captcha='valid' ∧ self.attempts<3)))

Acceptance test cases through structured misuse case description will be designed from the above mal-activity diagram SMC-SA-001. Test data is essential as it is used to execute test cases. We use the equivalence class partitioning technique to generate test data. We have two scenarios in the above activity diagram, and test cases will be designed for both scenarios. In the first scenario, users enter their username and Password to Login. IP of the system will be blocked for more than three invalid attempts from the same IP. When the login and password are valid and the user has made fewer than three attempts, the system will create a verification code. The User will enter the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

verification code that was delivered to the email address in the second scenario. If the code entered is authentic, the system will log the user in; otherwise, they must repeat scenario 1 until they obtain the code. The acceptance test scenarios for incorrect usernames, passwords, Captchas, and login attempts are shown in Table 3. For the second situation, we will now create a test case.

4.3. Acceptance test case Design

Acceptance test cases through structured misuse case description (SMCD) will be designed from the above mal-activity diagram shown in Figure 1. Test data is essential as it is used to execute test cases. We use the equivalence class partitioning technique to generate test data. We have two scenarios in the above activity diagram, and test cases will be designed for both scenarios. In the first scenario, users enter their username and Password to Login. IP of the system will be blocked for more than three invalid attempts from the same IP. When the login and password are valid and the user has made fewer than three attempts, the system will create a verification code. The User will enter the verification code that was delivered to the email address in the second scenario. If the code entered is authentic, the system will log the user in; otherwise, they must repeat scenario 1 until they obtain the code. The acceptance test scenarios for incorrect usernames, passwords, Captchas, and login attempts are shown.

For this situation, we will now create a test case. Username Valid Class: { A-Z},{ a-z } and Invalid Class: { 0-9}, {!@#$%^&*()[]{} }, Password Valid Class: { A-Z},{ a-z},{ 0-9},{ !@#$%^&*();:[]{} } and Invalid Class: { A-Z},{ a-z} Captcha Valid Class: {A-Z},{a-z},{0-9} and Invalid class: { !@#$%^&*();:[]{} } Login Attempts Valid Class:{0 < attempt =< 3 } and Invalid Class: {attempt > 3}.

In following Table 7, an acceptance test case is designed for structured Misuse case description.

Table 7. Acceptance test case Design.

TC #	Scenario	ECP	Input				Expected Output
			Username	Password	Captcha	Attempts	
TC-08	Invalid Username Unsuccessful login	Username: Invalid Class: {0-9, @#$%^&*()[]{} }	Admin123	Admin@98ml	As12	1	Unsuccessful Login due to the wrong username
		Password: Valid: {A-Z, a-z, 0-9, !@#$%^&*();:[]{} }		admin_#@!11			
		Captcha: Valid Class: {A-Z},{a-z},{0-9}			AB23C		
		Login Attempts: Valid: {0 < attempt =< 3 }				2	
TC-SA-12	Unsuccessful Login due to invalid Password	Username: Valid: { A-Z, a-z }	user				Unsuccessful Login due to the wrong Password
		Password: Invalid Class: Password = { A-Z},{ a-z}		1234			
		Captcha: Valid Class: {A-Z},{a-z},{0-9}			XYZ88		
		Login Attempts: Valid: {0 < attempt =< 3 }				3	
TC-SA-14	Unsuccessful Login due to invalid Captcha	Username: Valid: { A-Z, a-z }	ABC				Unsuccessful Login due to invalid Captcha
		Passsword:		Admin&12345			

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2023 doi:10.20944/preprints202305.0343.v1

		Valid: { A-Z, a-z, 0-9, !@#$%^&*();:[]{} } Captcha: Invalid class: { !@#$%^&*();:[]{} } Login Attempts: Valid Class: {attempt > 3}		ZX&12 1		
TC-SA-15	Unsuccessful Login due to more than three login attempts	Username: Valid: { A-Z, a-z } Password: Valid: { A-Z, a-z, 0-9, !@#$%^&*();:[]{} } Captcha: Invalid class: { !@#$%^&*();:[]{} } Login Attempts: Invalid Class: {attempt > 3}	User	Admin&123 ZXC12 4		Unsuccessful Login due to more than three login attempts
TC-SA-16	Successful Login	Username: Valid: { A-Z, a-z } Passsword: Valid: { A-Z, a-z, 0-9, !@#$%^&*();:[]{} } Captcha: Valid Class: {A-Z},{a-z},{0-9} Login Attempts: Valid Class: {0 < attempt =< 3 }	User	Admin&123 ZXC12 1		Successfully logged in

The designed test cases for the identified test combinations have been effective in detecting errors, while no other test combinations were found to be as effective other than those mentioned in MC/DC. This result has given us satisfaction with our approach, and we can confidently conclude that the test data generated for MC/DC is optimal. By following this process, we can ensure that the software we develop is robust and reliable, meeting the needs of our users.

5. Results and Discussion

In this research, we have constraints with four n values. Constraints specified in OCL can be with different n values. Test combinations identified are 5 for MC/DC and 16 for D/C. It is evident from the results that test data generated for MC/DC is optimal in each design experiment used. We performed a succinct analysis of the findings for n=1, n=2, n=3, and n=5. Additionally, we thoroughly specified the complete test data generation procedure for n=4 in the experiment given in the paper. It is important to note that for experiments involving n=2, n=3, and n=5, test data has been generated. Furthermore, it should be noted that no constraint was accessible for n>5. Based on the conclusions drawn from the findings of this study, a mathematical formula has been formulated for MC/DC and

 doi:10.20944/preprints202305.0343.v1

D/C approaches. The details of this formula are elaborated upon in the concluding section of this paper.

The following formula is derived for MC/DC on the basis of test data generated for all design diagrams where n=2, 3, 4, 5 respectively for each diagram.

$\sum t = n+1$ EQ 1

The following formula is derived for D/C on basis of test data generated for all design diagrams where n=2, 3, 4, 5 respectively for each diagram.

$\sum t = 2^n$ EQ 2

The test condition results clearly show MC/DC have generated optimal test data from constraint compared to Decision Coverage. As a next step, we have designed the test cases for optimal test conditions through equivalence class portioning.

5.3. Threats to Validity

The groups chosen for experimentation might differ before receiving any treatment. Because we have only utilized MC/DC and Decision Coverage to produce test data at the design level, it could be dangerous if suddenly Decision Coverage performed better when expressions became more complex or UML diagrams were altered. Additionally, if results are altered for higher constraint orders, i.e., more than 5, the formula obtained from the extracted data results may also alter.

6. Conclusion and Future Work

This research designs acceptance test cases for security threats through SMCD. In order to maximize test coverage through optimal test data generation we model security threat mitigation in SMCD. Mal activity diagram was designed from SMCD to generate consistent test data through MC/DC and D/C. A comparative analysis explains that MC/DC maximizes test coverage through optimal test data from SMCD in comparison to Decision Coverage. We have generated the test data from constraints for both MC/DC and Decision Coverage. MC/DC generated test data is n+1 while decision coverage is 2^n. Moreover, all the test data generated from the constraint is the same in each case, irrespective of the tester. Therefore, consistent test data will be generated through MC/DC or Decision Coverage due to constraint specification in Object Constraint Language. Thus reducing the test case subjectivity. In this experiment, regular expression orders up to order 4 are employed. In the future, it is possible to do it for constraint n>5 with higher complexity for other UML diagrams and other test data generation approaches can use MC/DC and D/C at the design level for extensive analysis.

Author Contributions: Conceptualization, T.H. and R.B.F.; data curation, T.H and R.B.F.; formal analysis, T.H. and R.B.F.; funding acquisition, M.A., G.S., R.A.; investigation, T.H., R.B.F., A.K and M.A.; methodology, T.H.; project administration, R.B.F., M.A., A. A., and G.S.; resources, T.H.; software, T.H.; supervision, R.B.F.; validation, T.H., R.B.F.,A.K, M.A. A.A.; writing original draft, T.H.; writing ,review and editing, T.H, R.B.F, R.A. and G.S . All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by Zarqa University, Jordan.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to extend their sincere appreciation to Zarqa University for supporting this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. M. Bharathi, "Hybrid Particle Swarm and Ranked Firefly Metaheuristic Optimization-Based Software Test Case Minimization," *Int. J. Appl. Metaheuristic Comput.*, vol. 13, no. 1, pp. 1–20, 2022, doi:

10.4018/ijamc.290534.

2. R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 33, no. 9, pp. 1041–1054, 2021, doi: 10.1016/j.jksuci.2018.09.005.
3. A. Jain, D. K. Tayal, M. Khari, and S. Vij, "Novel method for test path prioritization using centrality measures," *Int. J. Open Source Softw. Process.*, vol. 7, no. 4, pp. 19–38, 2016, doi: 10.4018/IJOSSP.2016100102.
4. M. Shahid and S. Ibrahim, "An Evaluation of Test Coverage Tools in Software Testing," *2011 Int. Conf. Telecommun. Technol. Appl.*, vol. 5, no. January, pp. 216–222, 2011, [Online]. Available: https://www.academia.edu/download/46300497/An_Evaluation_of_Test_Coverage_Tools_in_20160607-12431-14n7edq.pdf
5. S. M. Mohi-Aldeen, R. Mohamad, and S. Deris, "Optimal path test data generation based on hybrid negative selection algorithm and genetic algorithm," *PLoS One*, vol. 15, no. 11 November, pp. 1–21, 2020, doi: 10.1371/journal.pone.0242812.
6. X. Yao and D. Gong, "Genetic algorithm-based test data generation for multiple paths via individual sharing," *Comput. Intell. Neurosci.*, vol. 2014, 2014, doi: 10.1155/2014/591294.
7. C. Xue and R. D. Shawn, "A one-pass test-selection method for maximizing test coverage," *Proc. 33rd IEEE Int. Conf. Comput. Des. ICCD 2015*, pp. 621–628, 2015, doi: 10.1109/ICCD.2015.7357173.
8. M. Khari, A. Sinha, E. Verdú, and R. G. Crespo, "Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization," *Soft Comput.*, vol. 24, no. 12, pp. 9143–9160, 2020, doi: 10.1007/s00500-019-04444-y.
9. B. Lewis, I. Smith, M. Fowler, and J. Licato, "The robot mafia: A test environment for deceptive robots," *28th Mod. Artif. Intell. Cogn. Sci. Conf. MAICS 2017*, pp. 189–190, 2017, doi: 10.1145/1235.
10. M. Khari and P. Kumar, "An effective meta-heuristic cuckoo search algorithm for test suite optimization," *Inform.*, vol. 41, no. 3, pp. 363–377, 2017.
11. M. U. Khan, H. Sartaj, M. Z. Iqbal, M. Usman, and N. Arshad, "AspectOCL: using aspects to ease maintenance of evolving constraint specification," *Empir. Softw. Eng.*, vol. 24, no. 4, pp. 2674–2724, 2019, doi: 10.1007/s10664-019-09717-6.
12. S. K. Barisal, A. Dutta, S. Godboley, B. Sahoo, and D. P. Mohapatra, "MC/DC guided Test Sequence Prioritization using Firefly Algorithm," *Evol. Intell.*, vol. 14, no. 1, pp. 105–118, 2021, doi: 10.1007/s12065-019-00322-6.
13. M. Marré and A. Bertolino, "Unconstrained duals and their use in achieving all-uses coverage," *ACM SIGSOFT Softw. Eng. Notes*, vol. 21, no. 3, pp. 147–157, 1996, doi: 10.1145/226295.226312.
14. I. Rauf *et al.*, "The Case for Adaptive Security Interventions," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, pp. 1–52, 2022, doi: 10.1145/3471930.
15. I. Alexander, "Misuse cases help to elicit non-functional requirements," *Comput. Control Eng. J.*, vol. 14, no. 1, pp. 40–45, 2003, doi: 10.1049/cce:20030108.
16. H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empir. Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005, doi: 10.1007/s10664-005-3861-2.
17. P. Languages, P. Languages, D. Coverage, D. Coverage, M. Condition, and D. Coverage, "Comparison of DC and MC / DC code coverages".
18. H. Sartaj, M. Z. Iqbal, A. A. A. Jilani, and M. U. Khan, "A Search-Based Approach to Generate MC/DC Test Data for OCL Constraints," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11664 LNCS, no. August, pp. 105–120, 2019, doi: 10.1007/978-3-030-27455-9_8.
19. R. G. Tiwari, A. Pratap Srivastava, G. Bhardwaj, and V. Kumar, "Exploiting UML Diagrams for Test Case Generation: A Review," *Proc. 2021 2nd Int. Conf. Intell. Eng. Manag. ICIEM 2021*, pp. 457–460, 2021, doi: 10.1109/ICIEM51511.2021.9445383.
20. G. Robles, T. Ho-Quang, R. Hebig, M. R. V. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in GitHub," *IEEE Int. Work. Conf. Min. Softw. Repos.*, pp. 519–522, 2017, doi: 10.1109/MSR.2017.48.
21. M. El-Attar and H. A. Abdul-Ghani, "Using security robustness analysis for early-stage validation of functional security requirements," *Requir. Eng.*, vol. 21, no. 1, pp. 1–27, 2016, doi: 10.1007/s00766-014-0208-9.
22. S. Khamaiseh and D. Xu, "Software security testing via misuse case modeling," *Proc. - 2017 IEEE 15th Int. Conf. Dependable, Auton. Secur. Comput. 2017 IEEE 15th Int. Conf. Pervasive Intell. Comput. 2017 IEEE 3rd Int. Conf. Big Data Intell. Compu*, vol. 2018-Janua, pp. 534–541, 2018, doi: 10.1109/DASC-PICom-DataCom-

CyberSciTec.2017.98.

23. J. Hartmann, M. Vieira, H. Foster, and A. Ruder, "A UML-based approach to system testing," *Innov. Syst. Softw. Eng.*, vol. 1, no. 1, pp. 12–24, 2005, doi: 10.1007/s11334-005-0006-0.
24. Y. H. Wang and I. C. Wu, "Achieving high and consistent rendering performance of java AWT/Swing on multiple platforms," *Softw. - Pract. Exp.*, vol. 39, no. 7, pp. 701–736, 2009, doi: 10.1002/spe.
25. V. V. Ribeiro, "Understanding Factors and Practices of Software Security and," no. November, 2019.
26. J. J. Chilenski and S. P. Miller, "Applicability of modified condition/decision coverage to software testing," *Softw. Eng. J.*, vol. 9, no. 5, pp. 193–200, 1994, doi: 10.1049/sej.1994.0025.
27. M. W. Whalen, "The Effect of Program and Model Structure on MC / DC Test Adequacy Coverage," pp. 161–170, 2008.