Автономная некоммерческая организация высшего образования
«Университет Иннополис»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)
по направлению подготовки
09.03.01 - «Информатика и вычислительная техника»

GRADUATION THESIS
(BACHELOR'S GRADUATION THESIS)
Field of Study
09.03.01 – «Computer Science»

Направленность (профиль) образовательной программы
«Информатика и вычислительная техника»
Area of Specialization / Academic Program Title:
«Computer Science»

Тема / Topic	**Метаморфическое и Комбинаторное тестирование в сфере компьютерной безопасности** **Metamorphic and Combinatorial Testing in Cybersecurity**

Работу выполнил / Thesis is executed by	**Абрамов Матвей Евгеньевич** **Abramov Matvey**	подпись / signature
Руководитель выпускной квалификационной работы / Supervisor of Graduation Thesis	**Садовых Андрей Александрович** **Andrey Sadovykh**	подпись / signature

Иннополис, Innopolis, 2024

Contents

List of Tables

List of Figures

Abstract

The rapid development of software and its increasing integration into various aspects of our daily lives have brought about unprecedented challenges in ensuring its security. With the convergence of the non-IT and IT industries, the number of software errors and vulnerabilities has increased, posing significant risks to individuals, organizations, and society as a whole. Despite continuous efforts to improve cybersecurity, the omnipresence of cyber threats and the lack of a single foolproof method to ensure software correctness persist as pressing concerns.

This thesis project aimed to address the critical need for enhanced cybersecurity by investigating the potential of Metamorphic Testing and Combinatorial Testing as effective techniques to identify and mitigate software vulnerabilities. The research sought to evaluate the combined use of these methods and evaluate their efficiency, effectiveness, and comprehensive coverage in detecting security flaws.

To achieve the research objective, a rigorous study design was employed, focusing on the application of Metamorphic Testing and Combinatorial Testing to diverse software systems and scenarios. The study design incorporated various black-box testing techniques, including Domain Testing, Boundary Value Analysis, and the utilization of Metamorphic Relations to address the Test Oracle Problem. Extensive experimentation and analysis were performed to assess the benefits and limitations of this combined approach.

The research project yielded substantial findings, demonstrating that the integration of Metamorphic and Combinatorial Testing techniques significantly improved the efficiency and quality of software security testing. The approach exhibited superior coverage of potential vulnerabilities and offered a more comprehensive understanding of software behavior under various conditions.

The contribution of this work lies in providing a robust framework for improving cybersecurity through the synergistic use of Metamorphic and Combinatorial Testing. The results have direct applicability in industry and academia, offering a practical strategy to identify and address software vulnerabilities more effectively. This research contributes to advancing the state-of-the-art in cybersecurity and provides valuable insights into safeguarding the integrity and reliability of software systems in an increasingly interconnected world.

Chapter 1

Introduction

I Motivation

In recent years, the rapid evolution of software development has resulted in a significant surge in software errors and vulnerabilities. The escalating number of Common Vulnerabilities and Exposures (CVEs) reported each year reflects the growing challenges faced in ensuring the security and reliability of software systems [1]. As the digital landscape becomes increasingly complex and interconnected, the need for robust cybersecurity measures has become paramount.

The traditional approaches to software testing are often inadequate in detecting and mitigating vulnerabilities effectively. With the adoption of Continuous Integration practices as a standard in the industry, there's a heightened demand for automated testing methods that can seamlessly integrate into the development workflow. The primary objective is to identify and address bugs and vulnerabilities at the earliest stages of the software development life cycle.

Several techniques have been employed to enhance software testing and security, including but not limited to code analysis, fuzzing, and domain testing.

While these methods have proven to be valuable, there remains a pressing need for more efficient and effective approaches to bolster cybersecurity measures.

II Proposed Approach

This thesis proposes a methodology that combines two distinct testing techniques: Metamorphic Testing and Combinatorial Testing. The proposed approach adopts a semi-automated methodology for generating test cases. By leveraging both Metamorphic and Combinatorial Testing principles, the aim is to streamline the testing process while ensuring adequate coverage of the system under test. This hybrid approach strikes a balance between manual intervention and automated testing, optimizing the efficiency and effectiveness of the testing process.

One of the key advantages of the proposed method is its ability to achieve comprehensive test coverage with a relatively small number of test cases. Combinatorial Testing, by systematically exploring the interaction of input parameters, helps in reducing the number of test cases required. Meanwhile, Metamorphic Testing enhances the efficiency of the testing process by leveraging the inherent properties of the system.

III Terminology

This section presents an overview of key concepts in the field of metamorphic and Combinatorial Testing, focusing on their applications in cybersecurity. The definitions below lay the groundwork for understanding the methodologies discussed in subsequent sections.

A. Black Box Testing

Black Box Testing is a methodology where the internal workings or implementation details of the system under test are not known or considered by the tester. Testing is based solely on defined specifications or thought specs. Black-box testing relies only on the input/output behavior of the software [2].

Some examples of black-box testing techniques include:

- Metamorphic Testing
- Combinatorial Testing
- Fuzz Testing
- Boundary Value Analysis

B. Metamorphic Testing

Metamorphic Testing is an effective technique for alleviating the oracle problem, testing "untestable" programs where failures are not revealed by checking individual outputs, but by checking expected relations among multiple executions of the program under test [3].

Metamorphic Relation (MR) is a property or characteristic that should be maintained between multiple test case outputs. If an MR is violated, it suggests a potential defect in the software [4].

Metamorphic Group (MG) is a multiple sequence of inputs related to each other by a particular metamorphic relation. It consists of a sequence of source inputs and follow-up inputs [4]. *Example: MR1:* $f(x,y) = f(y,x)$ is a metamorphic relation, and *MG1: 1, 2* $\rightarrow$ *2, 1* is a metamorphic group.

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 1.1. Covering array for $f(a,b,c)$ with 2-way combinations. Notice, that each 2 columns contains all possible combinations of 0's and 1's.

C. Combinatorial Testing

Combinatorial Testing is a software testing technique that focuses on testing subsets of combinations of input values and preconditions to uncover defects. It's particularly effective when the number of parameters and their possible values is too large for exhaustive testing [5].

Consider a function $f(a,b,c)$ with boolean parameters.

Test Case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly [6]. For our function $f(a,b,c)$, a test case could be *a = 1, b = 0, c = 1*.

t-way combination is a set of t input values, one from each of t input parameters [6]. For our function $f(a,b,c)$, a 2-way combination could be *a = 1, b = 0*.

t-way covering array is a set of t-way combinations that covers all t-way combinations of input values. It is used to ensure that all combinations of input values are tested at least once. Fig. 1.1 shows a covering array for $f(a,b,c)$ with 2-way combinations.

D. Other Testing Techniques

Fuzz Testing is a technique involving providing invalid, unexpected, or random data as input to a computer program. The program is then monitored for exceptions such as crashes or failing built-in code assertions, or for finding potential memory leaks [7].

Boundary Value Analysis (BVA) is a software testing technique in which tests are designed to include representatives of boundary values. It is based on the idea that input values at the extreme ends of the input domain are more likely to cause errors in the system [8].

Property-based Testing is a software testing technique that involves checking whether a system satisfies a set of properties, which are logical assertions about the system's behavior. It is a form of black-box testing, where the tester specifies the properties that the system should satisfy, but does not specify how the system should satisfy them [9]. *Metamorphic Testing* is a form of property-based testing.

E. General Testing Terms

Oracle Problem is a problem in software testing where the tester does not know the correct output for a given input, and thus cannot determine if the system under test has passed or failed the test [5].

System Model is a model of the system under test (SUT) and/or its environment built from informal requirements, existing specification documents, or the SUT itself. It is used as input for automated test generation [5].

Test Suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviors.

F. Cybersecurity

Cybersecurity is the protection of computer systems and networks from information disclosure, theft of or damage to their hardware, software, or electronic data, as well as from the disruption or misdirection of the services they provide [5].

Cybersecurity Testing is for testing software system requirements related to security properties like confidentiality, integrity, availability, authentication, authorization, and non-repudiation. It involves techniques such as penetration testing, fuzz testing, static and dynamic analysis to identify vulnerabilities [5].

SQL Injection is a type of command injection vulnerability where the attacker can execute arbitrary SQL commands inside the application. This can lead to unauthorized access to the private data [10].

G. Test Metrics

Test Metrics are used to measure the effectiveness of the testing process. They are used to monitor the progress of testing, assess the quality of the software, and identify areas for improvement.

Test Coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. Some categorizations of test coverage are provided below:

- **Statement Coverage** measures the number of statements in the source code that have been executed by a test suite.
- **Branch Coverage** measures the number of if conditions, jumps, etc., in the source code that has been executed by a test suite.

- **Weak Coverage** means that the test suite has executed at least one of the possible paths through the code.
- **Strong Coverage** means that the test suite has executed all possible paths through the code.

IV Structure of the Thesis

This thesis is organized as follows:

- **Introduction**: Provides an overview of the research problem, objectives, and proposed methodology.
- **Literature Review**: Surveys the existing literature on software testing techniques, cybersecurity, and the principles of metamorphic and Combinatorial Testing.
- **Methodology**: Details the proposed approach, including the integration of metamorphic and Combinatorial Testing, and the semi-automated test case generation process.
- **Implementation**: Describes the implementation of the proposed methodology and presents the results of empirical experiments conducted to evaluate its efficacy.
- **Evaluation and Discussion**: Analyzes the findings of the experiments and discusses their implications for software testing and cybersecurity.
- **Conclusion**: Summarizes the key findings of the research and outlines potential avenues for future research in this area.

Chapter 2

Literature Review

Cybersecurity threats require innovative testing methods to ensure the reliability and security of the software. This literature review delves into the current state-of-the-art in Combinatorial Testing (CT) and Metamorphic Testing (MT) for cybersecurity, explores the integration of CT and MT, and assesses the efficiency of these methods working together and their potential applications in cybersecurity.

I Criteria and Process

The selection and analysis of literature for this review were guided by specific criteria and a structured process. Search keywords included "Combinatorial Testing in Cybersecurity," "Metamorphic Testing in Cybersecurity," and "Integration of CT and MT in Cybersecurity." The search was conducted across multiple academic databases, including IEEE Xplore, ACM Digital Library, and Google Scholar. Articles were chosen based on the number of citations, the date of publication, the popularity of the journal, and relevance to the topic. The method

involved a critical review of selected articles, focusing on their methodologies, findings, and implications in the context of cybersecurity testing.

II Combinatorial Testing in Cybersecurity

Within the cybersecurity domain, Combinatorial Testing has been effectively utilized, particularly in generating test cases aimed at identifying vulnerabilities like SQL injection [11] and XSS injections [12]. The focus of both studies was primarily on a singular type of vulnerability. These investigations have demonstrated that Combinatorial Testing is both rapid and efficient in uncovering vulnerabilities. However, it is important to note that the scope of these studies is somewhat restricted in terms of the range of technologies evaluated.

III Metamorphic Testing in Cybersecurity

The literature on Metamorphic Testing (MT) in cybersecurity provides insightful perspectives on its application and effectiveness. Segura *et al.* highlight the fundamental concept of MT to overcome the oracle problem in software testing, where the correctness of the output is ambiguous [3]. This aspect is particularly crucial in cybersecurity, where accurate results are essential but often difficult to define.

In their exploration of MT's application to cybersecurity, researchers have demonstrated its efficacy in detecting hidden bugs in vital applications. A notable example is the testing of compilers and code obfuscators, where traditional testing methods may fail [3] [13]. The use of MT in such scenarios underlines its ability to handle complex and critical software systems.

The development and implementation of the METRIC framework [14] mark a significant advancement in MT. This approach systematically identifies metamorphic relations (MRs) through a category-choice framework, simplifying the process of MR identification which was previously manual and ad hoc. This systematic approach is particularly advantageous in cybersecurity, where the generation of reliable test oracles is challenging. Iakusheva et al. showed that MR derivation can be further simplified by using domain specific guidelines [15].

MT's role in cybersecurity extends to various practical applications. It facilitates negative testing and the evaluation of security-related functionalities, often problematic with conventional testing methods. A case in point is the use of MT in identifying vulnerabilities such as the Heartbleed bug in OpenSSL, demonstrating its ability to handle complex real-world cybersecurity issues [16].

IV Integration of Combinatorial and Metamorphic Testing

The combination of Combinatorial Testing (CT) and Metamorphic Testing (MT) is explored by Niu *et al.* [6] in an innovative approach to software testing. Their study tackles the challenge in CT related to the creation of automated test oracles, a problem that often leads to manual and error-prone processes in testing.

Niu *et al.* introduce a new method called COMER, which blends MT with CT. This method focuses on forming pairs of test cases, known as Metamorphic Groups (MGs), based on specific metamorphic relations (MRs). These MGs allow for automatic checking of test results by verifying if the outputs go against the MRs.

V Conclusion

The exploration of Combinatorial Testing (CT) and Metamorphic Testing (MT) within cybersecurity, as outlined in this literature review, lays the groundwork for advancing the field of software testing. Although these methodologies have been examined separately in various contexts, their integration, especially in addressing cybersecurity challenges, opens new avenues for research. The innovative approach of combining CT and MT, as proposed by Niu *et al.* [6], demonstrates a promising direction towards automating and enhancing the accuracy of testing processes. This research seeks to build on these foundational studies, focusing on the implications and practical applications of integrating CT and MT in the domain of cybersecurity, a critical area that has not been fully explored in the existing literature.

Chapter 3

Methodology

This study employs the COMER methodology for the generation of test cases. In order to assess the efficacy of the framework and conduct comparative analyses with other existing frameworks, I undertook the task of implementing it from scratch using the Python programming language. The initial implementation by Niu *et al.* [6] was coded in Java. However, this implementation presented certain limitations; the code was compiled into a jar file, making it challenging to extend or modify functionalities. Consequently, I opted to develop my own implementation in Python, making necessary modifications to facilitate experimentation with different datasets and enhancing the usability of the framework. Python was chosen due to its user-friendly syntax and the availability of extensive libraries for data manipulation and visualization.

I COMER Framework

The COMER framework serves the purpose of generating test cases, leveraging both traditional Combinatorial Testing (CT) methods and incorporating

metamorphic relations during the generation process. Upon receiving inputs including a set of metamorphic relations, a set of constraints, and abstract input parameters, COMER generates test cases that adhere to the specified constraints while satisfying certain metamorphic relations.

The framework operates in two primary stages, namely:

A. Abstract Test Case Generation

The algorithm for abstract test case generation encompasses two main pathways. At each step, there exists a probability distribution determining the choice between two flows. In the first flow, the algorithm functions akin to a conventional CT generation algorithm, employing a greedy approach to generate test cases until achieving the desired t-way coverage. Conversely, the second flow involves selecting a previously generated test case and then generating subsequent test cases utilizing metamorphic relations. The second flow works by running Constrain Satisfaction Solver to produce follow up test cases giver previous test case, constraints, parameters and metamorphic relations.

B. Concrete Test Case Generation

Subsequently, the generated abstract test cases are mapped to concrete ones. For test cases generated via CT methods, the concrete test case is produced by randomly selecting a value from the domain corresponding to the abstract value. Conversely, for test cases generated through metamorphic relations, the framework utilizes the specified MRs to generate subsequent test cases. Full example of generating test cases from inputs is shown at Fig. 3.1

II Limitations

Despite the advancements made in the original COMER implementation, certain limitations were identified. Firstly, the reliance on a greedy approach for generating Combinatorial Testing (CT) test cases posed a notable drawback. While this method can increase the speed of generation process, it may result in an excessive number of test cases, potentially compromising efficiency and resource utilization. Secondly, the focus of the implementation solely on metamorphic relations with a single input and a single follow-up test case restricted its applicability to scenarios involving more complex MRs. Thirdly, the lack of provision for generating test cases for functions beyond those already supported in the implementation posed a significant constraint.

In my implementation, efforts were made to mitigate some of these limitations. To overcome the limitation of supporting only a limited range of functions, I worked to improve the framework by allowing the generation of test cases for any function. Additionally, in response to concerns regarding the effectiveness of the greedy approach, provisions were made to incorporate alternative algorithms for CT test case generation, thereby offering users flexibility in selecting the most suitable approach. However, it is important to note that the limitation pertaining to MRs with multiple inputs and follow-up test cases remains unaddressed in my implementation, representing an area for potential future research and improvement.

III Vulnerability Selection

This study utilizes the OWASP Top 10 vulnerabilities to assess the effectiveness of COMER framework in identifying and testing for common web application vulnerabilities. The OWASP Top 10 is a widely recognized list of the most critical security risks facing web applications [17].

Among the OWASP Top 10 vulnerabilities, I selected Injection as the primary testing scenario. Injection attacks involve injecting malicious code or data into an application's inputs in order to extract sensitive information, manipulate database queries, or execute arbitrary commands.

Injection is a particularly critical vulnerability due to its high potential impact and widespread prevalence in web applications. The OWASP Top 10 highlights injection attacks as one of the top 10 most critical security risks facing web applications. The goal of this paper is to test specifically SQL injections and general injection attacks, such as buffer overflow and input sanitization.

A. Test Case Generation

To test for SQL Injection vulnerabilities, I employed the COMER framework to generate test cases using existing combinatorial input space [10]. This approach involved generating a comprehensive set of input values that satisfied various metamorphic relations.

The generated test cases were designed to mimic real-world user inputs and simulate malicious injection attacks. The framework's ability to incorporate metamorphic relations enabled the generation of test cases that effectively targeted SQL Injection vulnerabilities.

B. Metamorphic Relations

The metamorphic relations employed in this study were designed to simulate various injection attack scenarios. They enabled the generation of different SQL statements by manipulating the input data, while the combinatorial part was used mostly to escape and bypass sanitization mechanisms.

These relations allowed me to manipulate the input data in such a way that different SQL statements could be generated, while still maintaining a realistic and comprehensive set of test cases.

IV App selection

For this study, I selected two web-based applications that are widely used in the field of penetration testing and vulnerability assessment. The chosen applications are:

- **WebGoat**: A web-based application security testing platform that allows users to simulate various types of attacks. [18]
- **DVWA (Damn Vulnerable Web Application)**: A deliberately vulnerable web application designed for testing and training purposes. [19]

These two applications were selected because they provide a comprehensive set of vulnerabilities and functionalities, making them suitable for testing the COMER framework. The use of these applications will allow me to demonstrate the effectiveness of the COMER framework in generating test cases that can identify and exploit various types of security vulnerabilities.

V Metrics

In order to measure the effectiveness of proposed approach different metrics were chosen:

- **Detection rate**: Measures the ability of the COMER framework to accurately identify and flag vulnerabilities within the selected web applications. This metric is fundamental in determining the effectiveness of the framework in identifying security threats.
- **Generation Time Efficiency**: Quantifies the speed and computational resources required by the COMER framework to generate test cases and identify vulnerabilities. This metric is essential for evaluating the practicality and scalability of the approach in real-world scenarios.
- **Number of test cases**: Similar to **Generation Time Efficiency**, but quantifies speed of test execution instead of generation. An excessively large number of test cases may also lead to increased testing overhead and resource consumption.

Several widely used metrics, like test coverage, were omitted due to the specifics of web security testing. In many instances, accessing the web application's code is not feasible. As a result, metrics suitable for a black box approach were utilized instead.

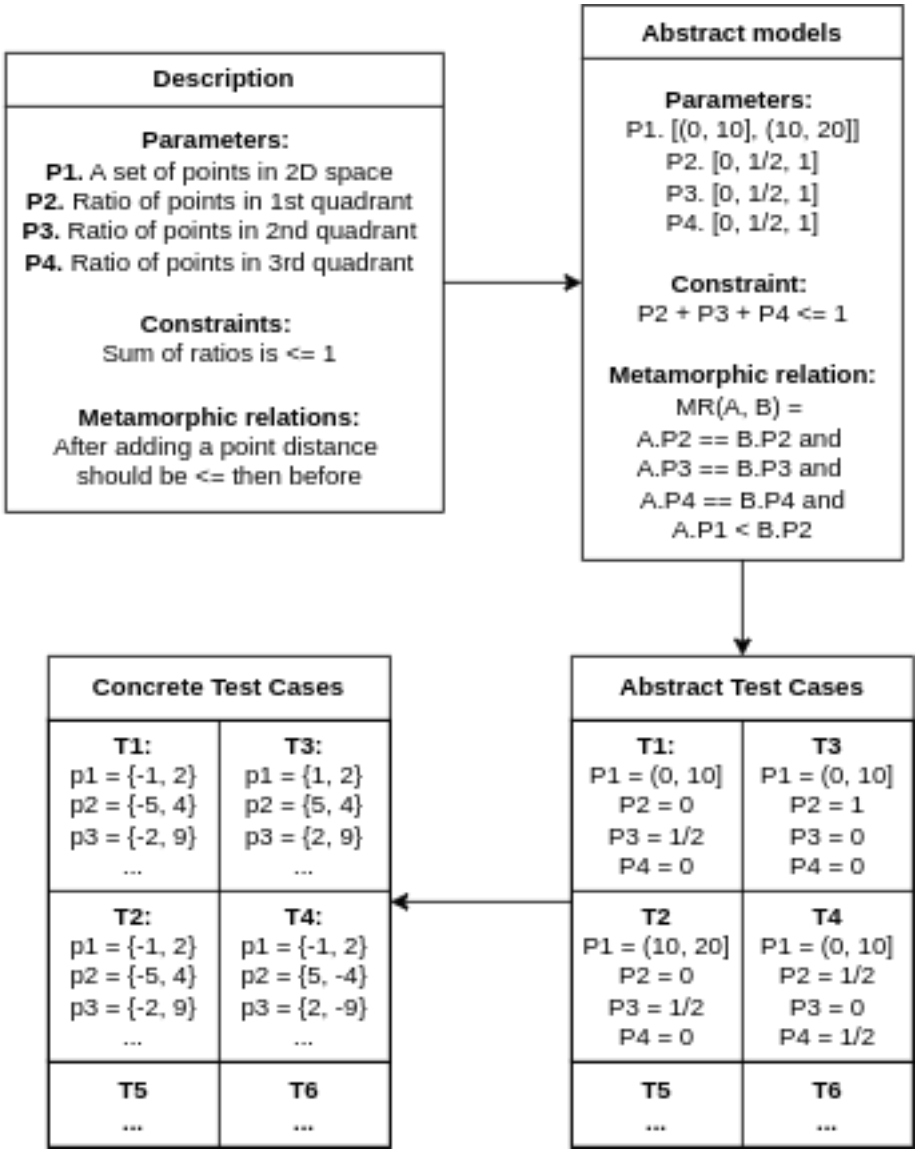

Fig. 3.1. Example of generating test cases for *FindClosest* function. Note that $T1$ and $T2$ are parameters generated by utilizing MR.

Chapter 4

Implementation

This section outlines how I created my version of the COMER framework. It presents the overview of solution, the example of usage, some intricacies of implementation and some thoughts on possible improvements.

I Details of Implementation

The implementation is encapsulated within a Python library, which can be seamlessly installed via the Python package manager, pip. This library facilitates testing procedures by requiring input parameters, constraints, Metamorphic Relations and producing concrete test cases. Library can be integrated with popular Python testing frameworks like *Pytest*, *unittest* or any major ones, given the simplicity of its usage.

Under the hood, implementation utilizes *python-constraint* package [20], which is a Constraint Satisfaction Solver written in Python. The choice was dictated by popularity of the tool and a lot of useful features. It supports different kind of constraints, including user-defined. It is also capable of using Back-

tracking, Recursive Backtracking and Minimum conflicts solvers. For my implementation, I decided to stick with Backtracking solver because it allows iterative generation of multiple solutions without solving the whole problem.

For the t-way generation I decided to use AllPairs method of generating tests. It has great efficiency and is easy to implement and extend, which is great for my use case [21]. Nevertheless, other methods could be used instead of AllPairs. The requirements for the method are:

- It should produce test cases which satisfy t-way coverage.
- It should be able to iteratively produce new test case given already existing ones.
- It should support specifying constraints in function form.

A. *Testing*

For testing the tool efficiency I needed a way interact with web applications. I decided to use Web Testing framework called *playwright* [22] which allows automation of web browser interactions via code. This tool is widely adopted in industry and has an integration with Pytest and Python in general. I used it to execute test cases produced by my library. Below is an example of injecting SQL via web form using playwright:

```
def sql(self, query: str) -> str:
    self.page.goto("http://localhost:4280/vulnerabilities/sqli_blind/")
    self.page.get_by_role("textbox").fill(query)
    self.page.get_by_role("button", name="Submit").click()
    pre = self.page.locator("pre")
```

```
        if self.page.get_by_text("There was an error.").is_visible():
            return ""
        if pre.is_visible():
            return pre.inner_text()
        return ""
```

B. COMER implementation

The main function responsible for generating test cases is as follows:

```
    def __next__(self) -> OrderedDict:
        r = random.random()
        if r > self._mr_probability and self.generated_cases:
            case_pre = random.choice(self.generated_cases)
            solution = self.csp_solver(case_pre)
            if solution:
                self.add_testcase_to_tested(solution)
                return solution

        return OrderedDict(super().__next__())
```

The function adheres to the foundational principles of the COMER framework. Specifically, based on a given probability, `self._mr_probability`, function chooses either a Combinatorial Testing (CT) or Metamorphic Testing (MT) based approach. In CT, it uses a Constraint Satisfaction Problem (CSP) solver to generate test cases iteratively. In MT, it selects an existing test case (`case_pre`) from the provided pool (`self.generated_cases`) and gener-

ates subsequent test cases using the CSP solver.

II Using framework

The example demonstrates how to use the framework for testing a web application. Suppose we want to test a web application that allows users to add bookmarks and smart tags to their accounts. We can define a set of parameters for our test cases as follows:

```
params = OrderedDict(
    {
        "highlight": [0, 1],
        "status_bar": [0, 1],
        "bookmarks": [0, 1],
        "smart_tags": [0, 1],
    }
)
```

Constraints are defined as a functions which accept test case as input and produce True or False indicating whether we should include particular test case into our domain. MRs are defined using test case as input and a new follow up as an output. In this particular example we do not define additional MR for generating concrete test cases for simplicity sake.

```
def constraint(highlight, status_bar, **kwargs):
    return highlight == status_bar
```

```
def mr(bookmarks, smart_tags, **kwargs):
    return OrderedDict({
        "bookmarks": smart_tags,
        "smart_tags": bookmarks,
        **kwargs
    })
```

After our domain is set, we can produce abstract test cases. In this example we use them as input to *Pytest* for producing parameterized test cases.

```
@pytest.mark.parametrize("testcase", Comer(params, constraint, MR(mr)))
def test_simple(testcase: OrderedDict):
    assert testcase["highlight"] == testcase["status_bar"]
```

Chapter 5

Evaluation and Discussion

This chapter presents the outcomes of test executions and compares it to the original Java COMER framework implementation as well as to CT approach. For this research, I formulated the following research questions:

- **RQ1:** How does the COMER framework compare to traditional Combinatorial Testing (CT) methods in terms of test case generation efficiency?
- **RQ2:** Can COMER be used to identify vulnerabilities in complex cybersecurity software more effectively than traditional CT methods?
- **RQ3:** Can this integrated technique be standardized into a framework or toolset for automated security testing?
- **RQ4:** Can relevant metamorphic relations and test pairs be systematically generated for complex cybersecurity software? Or does it require domain expertise?

I Evaluation

To evaluate the efficacy of my implementation of the COMER framework, I conducted a series of experiments comparing its performance with the original Java implementation by Niu *et al.* [6]. The experiments focused on assessing the framework's ability to generate test cases accurately and efficiently across various scenarios.

Charts below present the results of the experiments conducted on the COMER framework. The experiments were conducted on the following injection types: SQL Injection on DVWA, SQL Injection on Web Goat, Blind SQL Injection on Web Goat, and Form Sanitization in login forms. The injection were selected based on the large potential number of errors that could be uncovered and the complexity of the injection, including MR complexity and the number of test cases generated. Also, I choose to include Quicksort as a target for testing because original paper used it in its evaluation. All targets are described in Table I.

For the SQL injections, I used Metamorphic Relations (MRs) to generate test cases. The MRs were designed to simulate various injection attack scenarios, such as UNION injections, where we try to add additional rows to the resulting SELECT, Condition injections, where we try to modify condition check to always be True, and string escape, where we specifically try to escape strings. All of the relations were used to either add a constant to input or reverse some part of the input.

The results are shown in the three graphs, which present the number of test cases generated in Fig. 5.1, the number of faults detected in Fig. 5.2, and the execution time in seconds in Fig. 5.3. The results are considered for three different scenarios: COMER with 75% probability of selecting the MR approach,

COMER with 50% probability of selecting the MR approach, COMER with 25% probability of selecting the MR approach, and traditional CT approach using the AllPairs algorithm. The selection of such scenarios allows one to evaluate the difference between the traditional CT approach and the COMER framework, as well as the impact of the MR approach selection probability on the results.

TABLE I
Target Descriptions

Target	Domain size	MRs	Vulnerabilities
Quicksort	3^6	3	0
SQL DVWA	$2^8 * 3 * 5 * 8$	3	6
SQL Web Goat	$2^7 * 6 * 10$	3	6
SQL Blind Web Goat	$2^7 * 6 * 10$	3	6
Form Sanitization	$2^{11} * 4^2$	2	3

II Discussion

A. COMER and Traditional CT

To determine the efficiency of the approach, I compared the performance of the COMER framework with traditional CT methods in terms of test case generation efficiency. The results of the experiments demonstrate that the COMER framework is capable of generating test cases with a higher detection rate and about the same generation time compared to traditional CT methods. Different percentages of MR approach selection were tested to evaluate the impact of the MR approach on the results. The results indicate that the COMER framework can generate test cases more efficiently than traditional CT methods, especially when the MR approach is selected with a 50% probability.

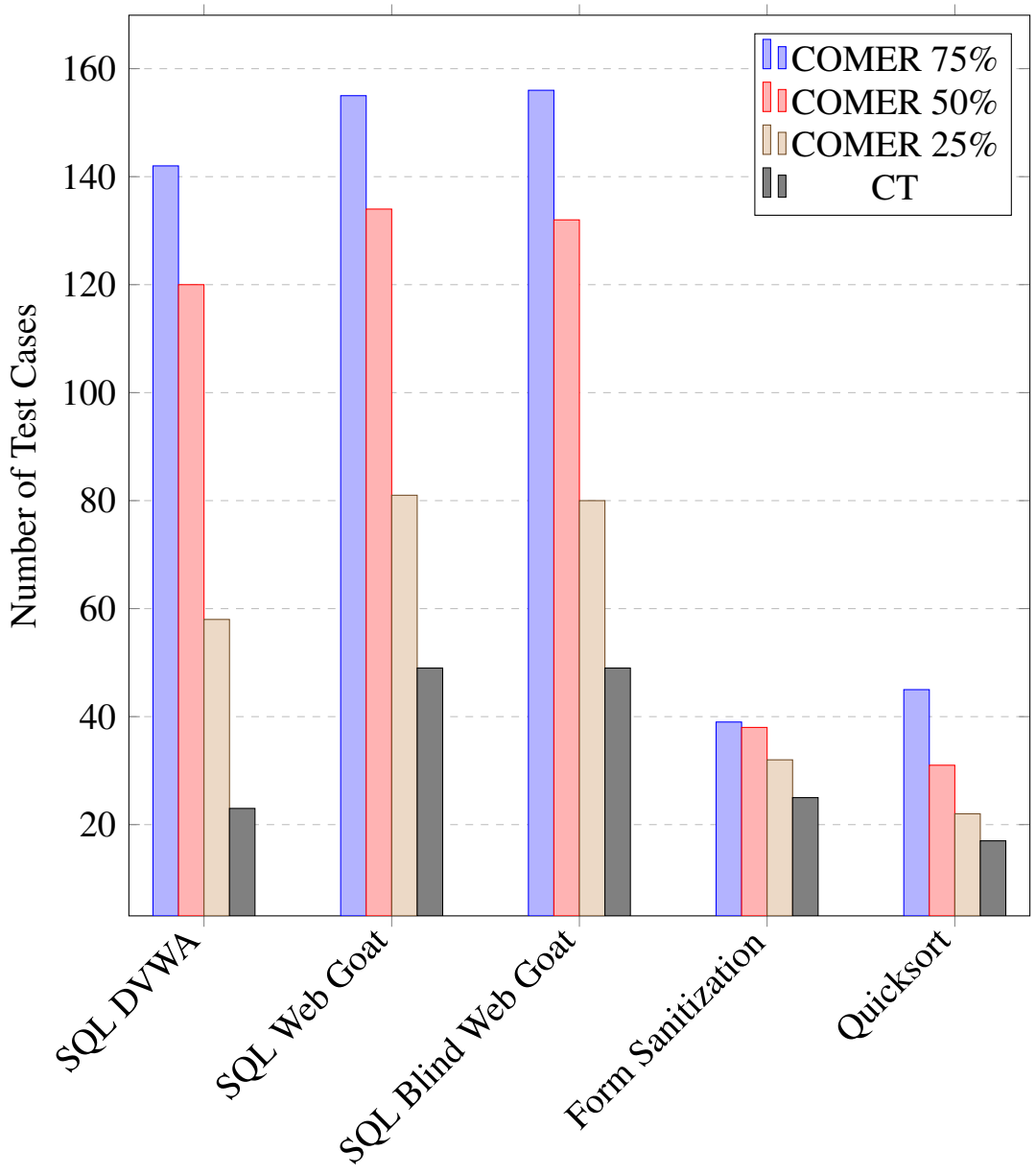

Fig. 5.1. Number of Test Cases generated for Different Injection Types. COMER Percentage Corresponds to probability that MR approach will be selected.

However, the number of test cases generated shown in Fig. 5.1 was higher than in the traditional CT approach, which may indicate that the COMER framework generates more test cases than necessary. This could potentially lead to increased resource utilization and inefficiency in the testing process. The results suggest that the COMER framework can be used as an effective alternative to tra-

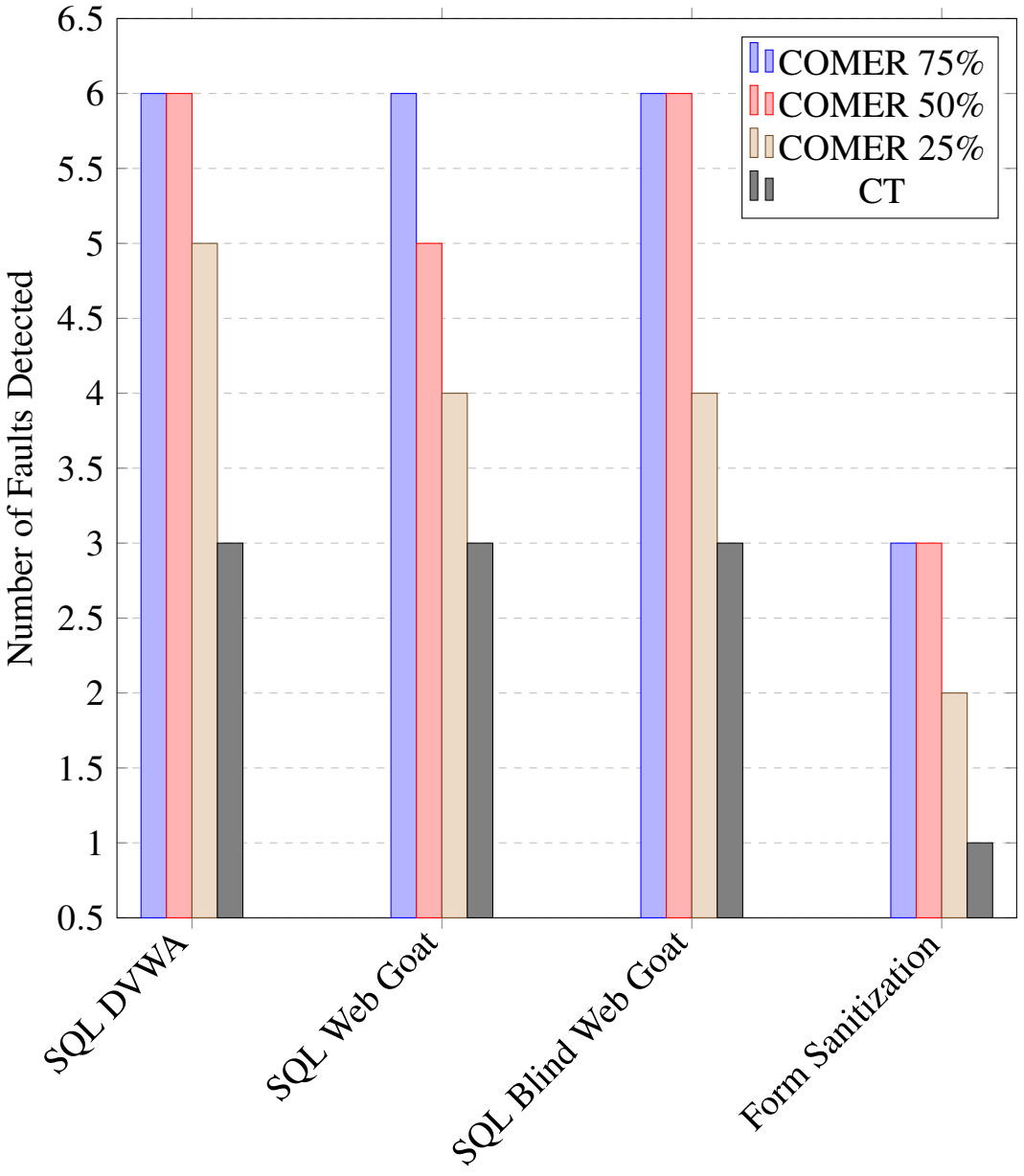

Fig. 5.2. Number of Faults Detected for Different Injection Types. COMER Percentage Corresponds to probability that MR approach will be selected.

ditional CT methods, but further optimization is needed to improve its efficiency. Currently, the utilization of the MR approach with 50% probability seems to be the most effective in terms of test case generation efficiency. On average, this method doubles the execution time, but it also doubles the number of detected faults.

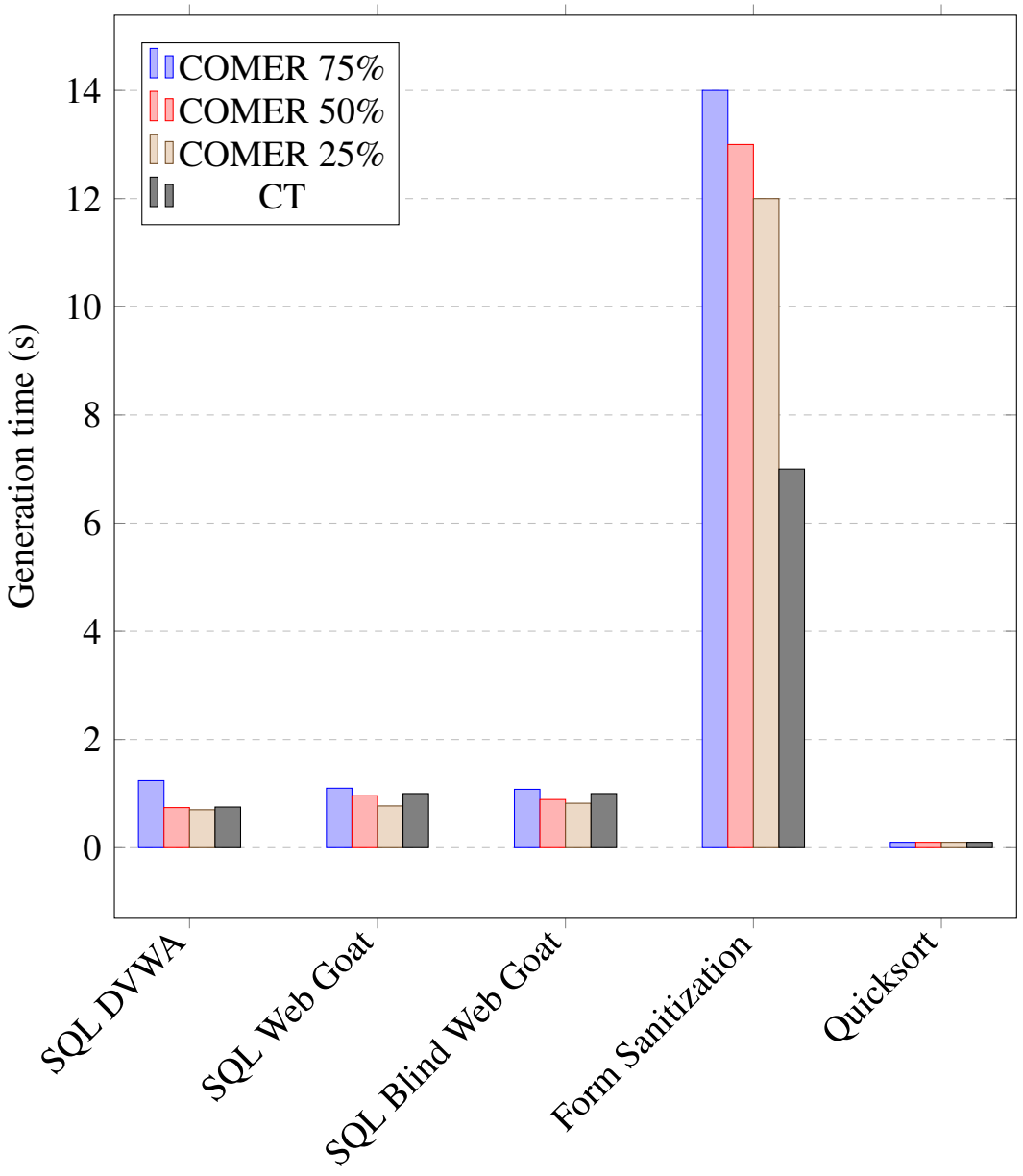

Fig. 5.3. Execution time in seconds for Different Injection Types. The *Form Sanitization* takes longer time due to larger domain. COMER Percentage Corresponds to probability that MR approach will be selected.

B. COMER in Cybersecurity

The results of the experiments demonstrate that the COMER framework is capable of generating test cases for testing Web Applications with a higher detection rate as shown in Fig. 5.2, and about the same generation time compared

to traditional CT methods, shown in Fig. 5.3. The use of MRs allows the framework not only find potential for SQL injections, but also execute different kinds of SQL injections and other vulnerabilities. The results indicate that the COMER framework can be used to identify vulnerabilities in complex cybersecurity software more effectively than traditional CT methods. The framework's ability to generate test cases that target specific vulnerabilities makes it a valuable tool for cybersecurity professionals seeking to enhance the security of their applications.

C. Standardization of COMER

Due to the way this evaluation was implemented, it makes it easy to test software not covered by this thesis. This indicates that the COMER approach can be standardized into a framework or toolset for automated security testing. The framework's ability to efficiently identify vulnerabilities and generate test cases makes it a valuable tool for cybersecurity professionals seeking to enhance the security of their applications. The result of my work can be used as a foundation for developing a standardized framework that can be applied across various domains and applications.

D. Systematic generation of Metamorphic Relations

The Metamorphic Relations (MRs) employed in this study were manually crafted for each specific test case. The process of fully automating MR generation remains a challenge due to the inherent complexity and domain knowledge it necessitates. The exploration of potential solutions, such as the utilization of Large Language Models (LLMs) or other methodologies, is out of the scope of this research. However, the results of this study demonstrate that the COMER

framework has the potential to be extended to support the systematic generation of MRs for complex cybersecurity software.

Chapter 6

Conclusion

This thesis has presented a comprehensive study on the integration of Combinatorial Testing (CT) and Metamorphic Testing (MT) for automated security testing of web applications. The proposed implementation of the COMER framework combines the strengths of CT and MT to generate test cases that can effectively identify vulnerabilities in complex cybersecurity software.

The evaluation of the COMER framework has demonstrated its effectiveness in generating test cases that can detect vulnerabilities in web applications. The results show that the COMER framework can generate test cases more efficiently than traditional CT methods, with a higher detection rate and similar generation time. However, preliminary tests of the current implementation showed that the execution time in some cases doubled compared to the traditional CT framework due to the increase in the number of test cases generated.

The study has also demonstrated the potential of the COMER framework to be standardized into a framework or toolset for automated security testing. The framework's ability to efficiently identify vulnerabilities and generate test cases makes it a valuable tool for cybersecurity professionals seeking to enhance the

security of their applications.

However, the study has also highlighted the need for more research in the area of systematic generation of Metamorphic Relations (MRs) for complex cybersecurity software. The manual crafting of MRs for each specific test case is a time-consuming and labor-intensive process that requires domain expertise. The development of methodologies or tools that can automate the generation of MRs would significantly enhance the effectiveness of the COMER framework.

I Future Work

The study has identified several areas for future research, including:

- Development of methodologies or tools that can automate the generation of Metamorphic Relations (MRs) for complex cybersecurity software.
- Investigation of the application of the COMER framework to other domains, such as mobile applications, IoT devices, other Cybersecurity vulnerabilities and domains.
- Exploration of the use of machine learning and artificial intelligence techniques to enhance the effectiveness of the COMER framework.
- Further improvements to current implementation to reduce the number of test cases generated and improve execution time.
- Compare with other state-of-the-art tools and frameworks for automated security testing.

These areas of research have the potential to further enhance the effectiveness of the COMER framework and to expand its applicability to other domains.

Bibliography cited

[1] *Cve statistics*, (accessed Mar. 18, 2024). [Online]. Available: https://www.cve.org/About/Metrics#PublishedCVERecords.

[2] M. Ehmer and F. Khan, "A comparative study of white box, black box and grey box testing techniques," *International Journal of Advanced Computer Science and Applications*, vol. 3, Jun. 2012. DOI: 10.14569/IJACSA.2012.030603.

[3] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Software*, vol. 37, no. 3, pp. 46–53, 2020. DOI: 10.1109/MS.2018.2875968.

[4] T. Chen, F.-C. Kuo, H. Liu, *et al.*, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, 4:1–4:27, Jan. 2018. DOI: 10.1145/3143561.

[5] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Chapter one - security testing: A survey," in ser. Advances in Computers, A. Memon, Ed., vol. 101, Elsevier, 2016, pp. 1–51. DOI: https://doi.org/10.1016/bs.adcom.2015.11.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245815000649.

[6] X. Niu, Y. Sun, H. Wu, *et al.*, "Enhance combinatorial testing with metamorphic relations," *IEEE Transactions on Software Engineering*, vol. 48, pp. 5007–5029, 2021. DOI: 10.1109/TSE.2021.3131548.

[7] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," Oct. 2018, pp. 2123–2138. DOI: 10.1145/3243734.3243804.

[8] F. Dobslaw, F. Neto, and R. Feldt, "Boundary value exploration for software analysis," Oct. 2020, pp. 346–353. DOI: 10.1109/ICSTW50294.2020.00062.

[9] D. MacIver, Z. Hatfield-Dodds, and M. Contributors, *Hypothesis: A new approach to property-based testing*, Nov. 2019. DOI: 10.21105/joss.01891. [Online]. Available: http://dx.doi.org/10.21105/joss.01891.

[10] D. E. Simos, J. Zivanovic, and M. Leithner, "Automated combinatorial testing for detecting sql vulnerabilities in web applications," in *2019 IEEE / ACM 14th International Workshop on Automation of Software Test (AST)*, 2019, pp. 55–61. DOI: 10.1109/AST.2019.00014.

[11] D. E. Simos, J. Zivanovic, and M. Leithner, "Automated combinatorial testing for detecting sql vulnerabilities in web applications," in *2019 IEEE / ACM 14th International Workshop on Automation of Software Test (AST)*, 2019, pp. 55–61. DOI: 10.1109/AST.2019.00014.

[12] B. Garn, I. Kapsalis, D. E. Simos, and S. Winkler, "On the applicability of combinatorial testing to web application security testing: A case study," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, ser. JA-

MAICA 2014, San Jose, CA, USA: Association for Computing Machinery, 2014, pp. 16–21, ISBN: 9781450329330. DOI: 10.1145/2631890.2631894. [Online]. Available: https://doi.org/10.1145/2631890.2631894.

[13] A. F. Donaldson, H. Evrard, A. Lascu, and P. Thomson, "Automated testing of graphics shader compilers," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, 2017. DOI: 10.1145/3133917. [Online]. Available: https://doi.org/10.1145/3133917.

[14] T. Y. Chen, P.-L. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 177–190, 2016, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.07.037. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121215001624.

[15] S. F. Iakusheva and A. S. Khritankov, "A systematic review of methods for deriving metamorphic relations," *Program Systems: Theory and Applications*, vol. 15, pp. 37–86, 2 2024. DOI: 10.25209/2079-3316-2024-15-2-37-86.

[16] T. Y. Chen, F.-C. Kuo, W. Ma, *et al.*, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016. DOI: 10.1109/MC.2016.176.

[17] *Owasp*, (accessed May. 11, 2024). [Online]. Available: https://owasp.org/www-project-top-ten.

[18] *Webgoat*, (accessed May. 11, 2024). [Online]. Available: https://owasp.org/www-project-webgoat/.

[19] *Dvwa*, (accessed May. 11, 2024). [Online]. Available: https://github.com/digininja/DVWA.

[20] *Python-constraint*, (accessed May. 11, 2024). [Online]. Available: https://github.com/python-constraint/python-constraint.

[21] *Pairwise efficiency*, (accessed May. 11, 2024). [Online]. Available: https://www.pairwise.org/efficiency.html.

[22] *Playwright*, (accessed May. 11, 2024). [Online]. Available: https://github.com/microsoft/playwright.