INTERNATIONAL JOURNAL OF EDUCATION AND INFORMATION TECHNOLOGIES
DOI: 10.46300/9109.2021.15.11 Volume 15, 2021

Exploring Software Security Test Generation Techniques: Challenges and Opportunities

Mamdouh Alenezi[1], Mohammed Akour[2], Hamid Abdul Basit[3]
[1,2,3]Computer Science Department, Prince Sultan University, Riyadh 11586, Saudi Arabia
[2]Information Systems Department, Yarmouk University, Irbid 21163, Jordan
Corresponding author: makour@psu.edu.sa

Received: April 23, 2021. Revised: May 25, 2021. Accepted: May 28, 2021. Published: June 3, 2021.

Abstract: Ensuring the security of the software has raised concerns from the research community which triggered numerous approaches that tend to eliminate it. The process of ensuring the security of software includes the introduction of processes in the Software Development Life Cycle where one of them is testing after the software is developed. Manually testing software for security is a labor-intensive task. Therefore, it is required to automate the process of testing by generating test cases by automated techniques. In this paper, we review various software security test case generation approaches and techniques. We try to explore and classify the most eminent techniques for test case generation. The techniques are summarized and presented briefly to covers all researches work that has been done in the targeted classification. Moreover, this paper aims to depict the sound of security in the current state of the art of test case generation. The findings are summarized and discussed where the opportunities and challenges are revealed narratively. Although the paper intends to provide a comprehensive view of the research in test case generation, there was a noticeable lack in the test case generation from the security perspectives.

Keywords—Security Test Generation Techniques, SLR, Techniques Weaknesses and strengths

I. Introduction

Raising the number of cyberattacks on software triggered major concerns of software security issues for the research community lately [3]. The well-known issues are attacks that manipulate the data, denial of service attacks, and cyber-attacks that reveal sensitive financial or other types of data [16]. A tremendous amount of effort has been made to enhance the security of software.

To ensure software security, it is been established to include processes to deal with security issues early in the software development lifecycle. Hence, this has initiated a new domain of research called secure software development. The secure design ensures secure software development. Designing security in software has become the best practice for developing secure and trustworthy software in a cost-efficient manner. Security design does not ensure security in implementation as there are still possibilities of vulnerabilities that become part of software during the implementation process. To ensure secure implementation design level artifacts are used for testing the implementation. Security tests are generated from design-level artifacts. Security attacks are usually initiated by providing invalid inputs that are chosen by the attacker. The challenge comes because of the complex nature of input space, which makes testing of programs with invalid inputs very difficult [48][49]. Therefore, there is a dire need for automating the process of generating security test

cases. The process of detecting security vulnerabilities is automated with cost-effective testing techniques. Thus in the literature, we can find various efforts for automating software security test case generation techniques [3][53]. Research on test data generation is going on since 1970 [23][40][47]. The automatic test case generation has received a considerable amount of attention from researchers. Therefore, in the literature, we find a large number of heterogeneous techniques. There is a need to critically review existing software security test case generation techniques to look for open problems and sketch the future of test case generation techniques.

In this paper, we compile and discuss various software security test case generation techniques found in the literature. In addition, we identify research challenges and future direction. The paper is organized as follows. Section 1 contains the introduction. In section 2, we discuss the classification of automated software test case generation techniques into two major categories: type-based software test case generation techniques and Algorithm-based software test case generation techniques. Section 3, presents type-based software test case generation techniques. In section 4, we discuss algorithm-based software test case generation techniques. Section 5 focuses on security test case generation research work. Section 6, presents our analysis of the literature under the heading of discussion and findings. In section 7, we conclude the paper.

II. RESEARCH METHODOLOGY

The main goal of this research paper is to explore the available software test case generation techniques and briefly manifest the mechanism for each technique. Moreover, we highlighted the test case generation where the security attribute was the essence behind the test generation. We compare these techniques by considering the challenges and opportunities of the studied techniques.

The methodology steps in this review are as follows: 1. The authors articulate the review protocol. 2. Carry out the review (search, list and evaluate primary studies, extract and synthesize data to produce a concrete result). 3. Analyze the findings. 4. Report the findings. 5. Discuss the findings. In the review protocol, we describe the research questions to be addressed in this work. Moreover, we identify the set of databases that are searched and specify the methods used to identify, assemble, and assess the evidence. To minimize the researcher bias, the protocol is developed by the first author, reviewed by the second author, and then finalized through the group discussion.

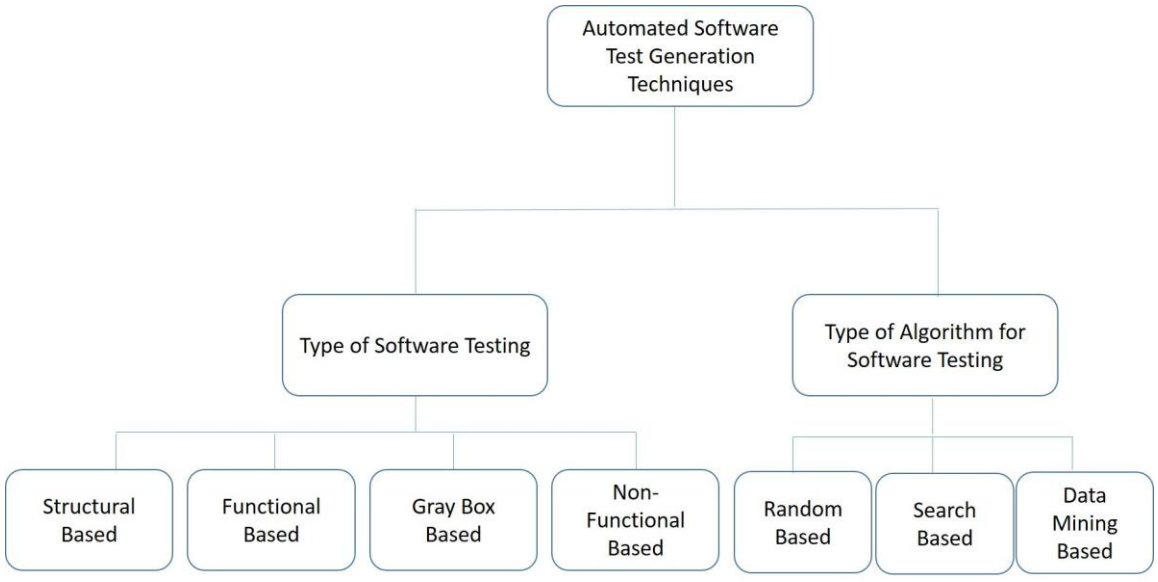

Fig.1 Classification of Automated Software Test Techniques

A. *Research Questions*

The aims of this study are twofold. First, we would like to explore and expose the diverse test case generation in the literature. Second, we would like to find out if there are techniques that take security as the main purpose of the test case generation or not. The high-level question addressed by this review is: What types of techniques and approaches for software test case generation can be found in the literature. This high-level research question is subdivided into three specific research questions to better guide the literature review process.

RQ 1: What are the common taxonomies of test case generation techniques?
RQ 2: Are there security-oriented test case generation techniques in the scientific literature?
RQ 3: What are the limitation and the opportunities provided by the selected studies?

B. *Database Source Selection And Search Criteria*

As a preliminary step and before starting looking for research papers, databases must be carefully chosen to improve the possibility of obtaining the most broad and relevant resources. In this review, the following criteria are used to select the source databases: The database must include journals and conference proceedings that cover: test case generation, software security, automatic and manual test case generation, test case prioritization, and empirical studies. To minimize the redundancy of journals and proceedings across databases, the list of databases is reduced where possible.

Authors determine the keywords that should be used to find any articles that can provide an answer to the research questions as follows: (approach or method or methodology or technique) AND ((“Test Case generation”) OR (“Security testing”)). Therefore, the search string was "Culture" AND "DevOps".

The search approach includes search resources and search steps as follows:

In order to find the sources in the literature about test case generation and security testing, the search was conducted on four common digital databases: 1- ACM Digital Library 2- IEEE Xplore Digital Library, 3- ScienceDirect, 4-Wiley Online Library. The search steps are presented in Figure 2.

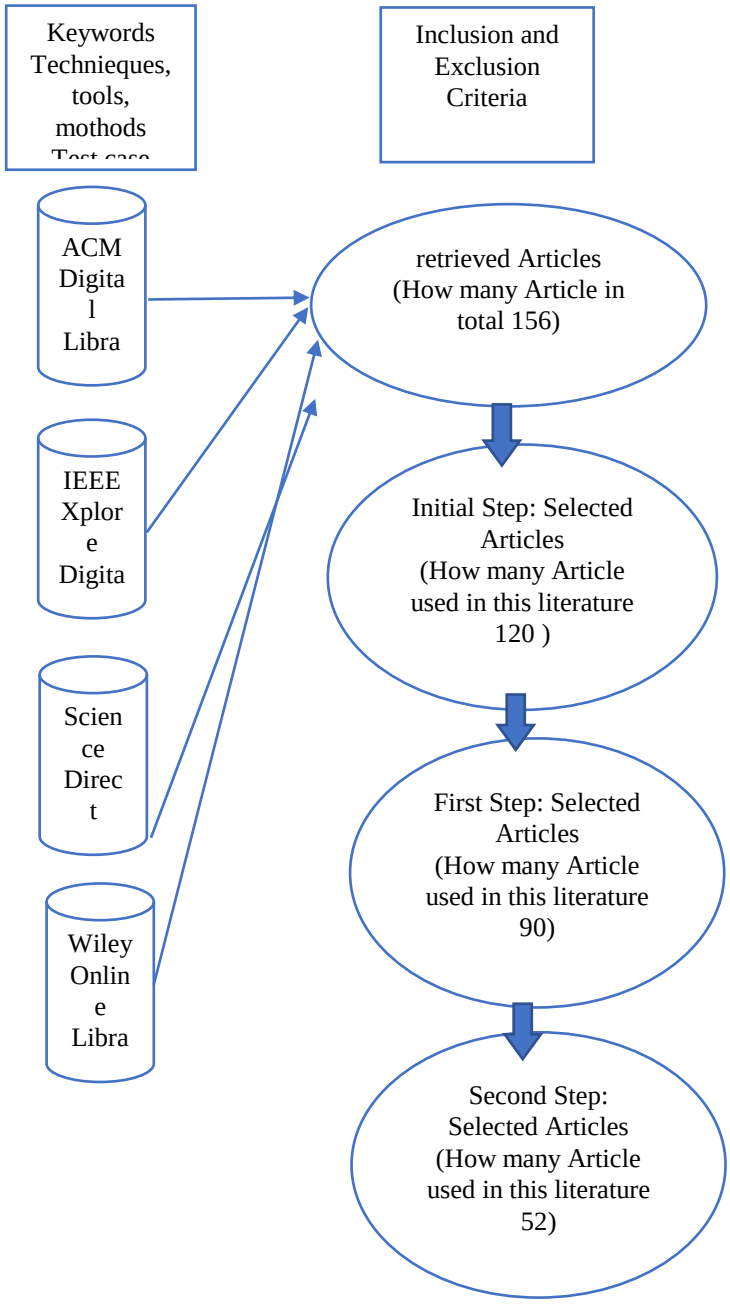

Figure 2. Search Steps

The search strings were used and the retrieved articles 156 papers. In the Initial step, we examined the titles, abstracts, and keywords of the articles where the irrelevant papers are removed and shortlisted the papers to 120.

In the first step, all remained papers were checked based on the full text, we produced two classifications here:
Relevant papers: where the papers are relevant to the study.
Irrelevant papers: where the papers are not relevant to the study, we exclude these papers.

Whenever authors have a uncertainty or conflict about the relevancy of a paper, we pushed the paper to be in the relevant list and make the final decision to exclude the paper or leave it during the second step when the full texts of the papers were studied again. The remaining papers were 90.

In the second step, we examined the remaining papers against inclusion and exclusion criteria as shown in Table 1. The final relevant papers were 52.

To make sure that the inclusion and exclusion criteria were consistent, authors select some of the final list papers randomly and re-evaluate the relevancy of these papers. Still, completeness is not guaranteed in this study as we may not find other relevant papers. However, to the best of our knowledge, this is the first SLR to address the security test case generation techniques.

Table 1 – Inclusion and Exclusion Criteria

Exclusion Criteria	Inclusion Criteria
Papers that are developed only on scholar judgment	Articles address software security testing and test case generation.
Posters, short papers**,** tutorials	Papers navigate security test suite generation, prioritization, or reduction
Studies that are not directly related to any of the addressed research questions	Papers explore the challenges, advantages, disadvantages of the current test case generation in terms of software security.
Pre-review conference or journal papers	Empirical studies (qualitative or quantitative)
Studies that are written other than the English language	
Studies providing unclear and misleading findings.	

C. Data Extraction

All papers from the second step are documented and the following data is extracted: 1) DB (Source), 2) Paper Title, 3) Authors, 4) Publication year, 4) Test Classification, (vii) paper abstract. The test case generation classifications are extracted initially from the paper title, abstract, or the introduction of each selected paper, Table 3 presents a set of taxonomies that were depicted and assigned to the papers. The process of classification and paper assignment was further refined, sometimes we used sticky notes to record the classification and paper assignment.

D. Research Quality Assessment

After assuring the relevancy of the papers, we performed the quality assessments of the selected papers to make sure that only high-quality papers are considered in this study. The quality assessment criteria are presented in Table 2. Each quality assessment criterion is evaluated as yes or no for each paper, and papers with only yes answers for all the criteria were included in the final list. (How many papers- used exactly in this study 156) of the (how many papers – the result of the second step 52) selected studies satisfied the quality questionnaire.

TABLE 2 – QUALITY ASSESSMENT CRITERIA

S. No	Quality Assessment Criteria
1	Is the paper based on a strong research methodology and is not just a report promising experimental evaluation?
2	Does the paper present clear goals of the research?
3	Does the paper present an appropriate description of the paper content?
4	Does the research methodology describe well-designed steps that navigate the main contribution of the research?
5	Is the research methodology sound and appropriate in terms of the main contribution of the research?
6	Is the collected data (if any) are commonly used by researchers?
7	Does the paper build a clear result and lessons learned?

III. RESULT AND DISCUSSION

This section presents the overview answers for the addressed research questions. The main contributing studies are 52 out of the initial collected set 120.

A. RQ 1: What Are The Common Taxonomies Of Test Case Generation Techniques.

After the full review of the prime studies, the final taxonomies were developed after conducting the data extraction and agreement steps presented in the previous section. Table 3 summaries the main taxonomies, where column 2 presents the main taxonomy, Column 3 presents the sub-taxonomies.

TABLE 3 – STUDY MAIN TAXONOMIES

ID #	Main Taxonomy	Sub-taxonomy
1	Type-Based testing	structural
		functional
		non- functional
		gray box
2	algorithm-based	randomized
		search-based
		data mining techniques

Below we discuss the classification of automated software test case generation techniques as found in the literature. Testing techniques can be classified along two main dimensions: one based on the type of testing and the other based on the type of algorithm used. The testing type-based classification consists of four types of techniques: structural, functional, non-functional, and gray box testing techniques. The algorithm-based classification consists of randomized, search-based, and data mining techniques.

These testing technique classifications are described as follows.

i. TYPE BASED CLASSIFICATION

- Structural: Test cases are generated with the help of a control flow graph or system source code. With a structural testing test, adequate criteria are covered.
- Functional: System specification is used to generate test cases. The functionality of software under test is tested.
- Non-functional: Working in the system is tested in this type of testing. The tests measure characteristics of system and software which is quantified.
- Gray Box: Both structural and functional information is used to generate test cases.

ii. ALGORITHM-BASED CLASSIFICATION

Randomized: Event sequences and test cases are generated randomly. The best test set for the problem is generated.

Search-Based: Test case generation is considered an optimization problem.

Data Mining: Input/Output of a program under test is analyzed. The number of test cases is reduced by factoring out unimportant and infeasible test cases.

Type-Based Software Test Case Generation Techniques

In this section, we will discuss the type-based software test case generation techniques found in `the literature.

i. STRUCTURAL BASED SOFTWARE TEST CASE GENERATION TECHNIQUES

The authors in [52] stress the automation of the security testing process. An approach for generating security tests with the use of formal threat models is presented. Attack paths are generated from threat models. The paths are transformed into executable test code. The approach is applied to two real-world systems Magento and Filezilla. Experiments demonstrate security tests we able to find several security problems in these systems.

ii. FUNCTIONAL BASED SOFTWARE TEST CASE GENERATION TECHNIQUES

In [15] motivated with automated validation of software systems. The authors used test case generation to enhance dynamic specification mining. It is claimed that this work is the first work towards a combination of a systematic test case and typestate mining. In an experiment, eight hundred errors were scattered into 6 Java projects, the static typestate verifier reports more true positives and fewer false positives.

The authors in [51] addressed the research on the automatic generation of system test cases from natural language (NL) requirements. Existing proposed approaches require manual intervention. The authors proposed Use Case Modeling for System Test Generation (UMTG), which enables automatic generation of system test cases from use case specification and domain model. Motivated by the accepted practice of use case specification and domain modeling, the proposed approach adopts the practice. At the same time, behavioral modeling is considered a difficult and expensive exercise. The feasibility of the approach is demonstrated by an industrial case study in the automotive domain.

In [27] the authors proposed a framework WebSob for automated testing of web service. Manually testing web service is a tedious job therefore there is a need for an automated tool to test web service. Tool for automatically generating and executing web-service requests with the analysis of subsequent request-response pairs. At first, the necessary Java code is automatically generated which implements a client which is a service requestor. Unit tests are generated using automated unit test generation tools where the executed unit tests invoke the service. Request-response pairs from web service invocation were analyzed and robustness problems were identified.

The authors in [7] present the result of a case study of generating test cases for a fragment of smart card GSM 11-11 standard. Contribution of testing environment B-Testing-Tool in the industrial process on the real-life application is carried out. Generated test sequences are compared with great quality manually designed tests. The approach is validated with this comparison.

In [54] the authors extend metamorphic testing into a user-oriented approach of software quality assessment, validation and verification. 4 search engines are studied Bing, Google, Chinese Bing and Baidu. Results from the study show that users and search engine developers can get benefit from this study. It is demonstrated that the proposed approach reduces the oracle problem.

The authors in [13] review the existing research on metamorphic testing and present challenges that need to be addressed. Further improvement in metamorphic testing and opportunities for novel research are discussed.

In [37] the authors address mutation analysis and propose a path selection strategy for selecting test cases that kill mutants. 55 million program paths were investigated which is based on a strategy to reduce the effects of infeasible paths.

iii. Gray Box Based Software Test Case Generation Techniques

The authors in [45] present a survey that improves the understanding of UML-based testing techniques. Authors considered test case that was generated from: state-chart, sequence and activity diagram. Moreover, many research techniques that are based on, graph theory, formal specifications, heuristic testing and direct UML specification processing are classified.

iv. Non-Functional Based Software Test Case Generation Techniques

In [21] automated security validation system AppInspector is proposed. The proposed system analyzes apps and generates reports of security and privacy violations.

The authors in [26] propose an automated test generation for access control policies. The manual generation of the test is tedious and manual tests are not considered enough to employ numerous policy behaviors. A novel framework is put forth along with that supporting tool known as CIRG (Change Impact Request Generation) is used to generate tests based on change impact analysis. CIRG efficiently generates tests that help to gain high structural coverage policies and surpasses random test generation.

[41] proposed input generation techniques for android based operating system apps. The techniques differ from each other in the way the inputs are generated and the strategy used to investigate the performance of the studied apps. A comparison between the existing test input generation techniques is conducted to summarize the opportunities and the limitations of the studied approach. Furthermore, the effectiveness of these tools is evaluated and the techniques employed with respect to four metrics: the ability to work on multiple platforms, ease of use, ability to detect faults, and code coverage.

Algorithm-based Software Test Case Generation Techniques

In this section, we will discuss the algorithm-based software test case generation techniques found in the literature.

i. Random Based Software Test Case Generation Techniques

In [20] the authors studied the effectiveness of automated test generation techniques that produce coverage providing tests. Their effectiveness in terms of fault detection ability is not studied adequately. Effectiveness of test suites that satisfy four coverage criteria with the help of counterexample-based test generation and random generation approach. Results yield three conclusions. Coverage criteria are a poor sign of fault-finding effectiveness. Secondly, the use of structural coverage as a supplement instead of a target for test generation has a positive impact. Random test suites are reduced to coverage providing subset detecting up to 13.5% more faults than test suites. Thirdly, observable Modified Condition/Decision Coverage (MC/DC coverage), criteria that are designed to account for program structure and selection of test oracle caters the failings of traditional structural coverage criteria. The results indicate risks inherent in expanding test automation in critical systems along that highlights the areas of research in automating tests.

ii. Search-Based Software Test Case Generation Techniques

The authors in [19] propose an automated technique of software testing. The automated process helps in the identification of bugs and errors. There are various meta-heuristic techniques employed in removing bugs. The artificial bee colony technique is one of them. The algorithm intelligently synchronizes bees, which helps to discover nodes in software code. This paper critically reviews the technique. The proposed approach is scalable and requires less computation time.

In [44], a novel approach to generate test paths is proposed. By using the standard Unified Modeling Language (UML), Activity diagram, and Activity dependency table (ADT) test paths are generated. Test paths are prioritized with the use of the Tabu search algorithm. The prioritized paths are used in regression testing, system testing, and integration testing. The efficiency of the test scenario is evaluated using a cyclomatic diagram.

The authors in [18] propose a research prototype EvoSuite, which automatically generates test suites for classes written in the Java programming language. Technical challenges that EvoSuite should address which Java classes coming from open source project should handle.

In [28] a survey of meta-heuristic search techniques applied in automating test data generation for structural and functional testing is presented. Future directions of research in each of heterogeneous individual areas are discussed.

The authors in [1] discussed various search heuristics which are based on OCL constraint. These techniques lead to test data generation and automate Model-Based Testing in industrial applications. The feasibility of the proposed approach is evaluated using empirical analysis.

In [29] the authors proposed an approach where input from the internet is pursued. The program identifiers are restructured into web queries. The URLs are downloaded, split into tokens, and then seed search-based data generation techniques. Empirical evaluation is carried out with string input validation code from 10 open-source projects. Valid string inputs were retrieved from the web with 96% of heterogeneous string types analyzed. With the approach, coverage was improved for 75% of Java classes with an average increase of 14%

The authors in [25] propose a security test case derivation from design-level artifacts. An empirical study to show the feasibility of the approach.

In [17] a novel paradigm is proposed where whole test suites are evolved to cover all coverage goals. The total size is kept small at the same instant. The approach is implemented in EvoSuite. Open-source libraries and industrial case studies of 1741 classes EvoSuite achieved 188 times the branch coverage of the conventional approach which targets a single branch. 62% of smaller test suites were used in the evaluation.

iii. Data Mining Based Software Test Case Generation Techniques

The authors in [2] proposed an algorithm based on concolic testing which generates a sequence of events automatically and systematically. The path explosion problem is removed. The approach is implemented in android. Results demonstrate the effectiveness of five android applications.

In [35] novel method of generating test cases for black-box autonomous systems. As a result, a method of searching for challenging scenarios of the autonomous system under test is put forward. Adaptive sampling which intelligently searches state space for test scenarios is also introduced. With the use of unsupervised clustering techniques scenarios are grouped according to the performance modes.

In [32] the authors propose a framework that automatically generates a test case for Javascript applications. The approach called JSEFT combines function coverage maximization and function state abstraction algorithms that efficiently generate test cases. Evaluation is performed by using 13 Javascript-based applications which demonstrate that generated test cases achieve a coverage of 68% and JSEFT detects Javascript and DOM fault with high accuracy. JSEFT outperforms the current Javascript test automation framework with respect to coverage and detected faults.

The authors in [22] present two phases approach for discovering event sequences that find the targeted line of code in the application. The experimental studies on numerous android applications demonstrate the technique was able to produce the event sequences successfully.

In [8] propose a novel technique for alleviating traversed code paths by discarding the ones having side effects similar to a previously explored path. A mix of open source applications and device drivers decreases the number of paths traversed.

The authors in [39] present a white-box framework of testing deep learning framework DeepXplore. Incorrect corner case behaviors in state-of-the-art deep learning models with thousands of neurons that are trained with five popular datasets which include imagenet and udacity self-driving challenge data.

In [12] the authors discussed the state of the art in dynamic symbolic execution. The contribution of this work is fivefold. The first theoretical foundation of dynamic symbolic execution is summarized. Secondly, the challenges of turning ideas into reality are presented. State-of-the-art solutions with advantages and disadvantages for the challenges is discussed. Twelve typical tools were analyzed. Finally, future research directions are presented.

The authors in [14] propose a framework of dynamic tainting which is flexible and customizable. Traditionally dynamic tainting cannot be extended and adapted to a new context. Data flow and control flow-based tainting can be performed. The framework is called DYTAN. An implementation of the framework which works on x86 executables along with initial studies demonstrates how DYTAN is used to implement the different tainting-based approach. It is demonstrated how DYTAN is used in real software i.e. Firefox. Specific characteristics of the tainting approach can affect the efficiency and accuracy of taint analysis.

In [34] novel method of discovering integer bugs with the use of dynamic test generation on x86 binaries was presented. The method is implemented in a prototype tool SmartFuzz which is used to analyze Linux x86 binary executables. A reporting service metafuzz.com for reporting bugs discovered by SmartFuzz and black-box fuzzing tool zzuf. metafuzz.com recorded more than 2614 test runs with 2361595 test cases. Experiments discovered 77 total bugs in 864 compute hours which cost an average of $2.24 per bug considering current EC2 rates.

In [9] present a technique that utilizes code to automatically generate a test case at run time. This is done with a combination of symbolic and concrete execution. The technique was applied to real code and discovered numerous errors i.e. simple memory overflow and infinite loops.

In [30] presents a genetic algorithm for the automated generation of the test. The work is an extended form of previous research on dynamic test data generation. GA-based approach was implemented and its effectiveness is demonstrated for several programs.

The novel approach of testing web services based on data perturbation is presented in [36]. XML messages are modified based on rules defined on message grammars which are used as tests. Two methods are used by data perturbation for testing web service: data value perturbation and interaction perturbation. Experiments demonstrate the usefulness of the proposed approach.

The authors in [50] propose a systematic technique called scope bounded testing which develops novel specifications for effectively generating tests for products in the software products line. Experimental results with the use of heterogeneous data structure products demonstrate speedup over conventional techniques.

In [31] the authors present an approach that combines the approach of testing web applications by writing test cases in SELENIUM and using a crawler to explore dynamic states of the application. The proposed approach mines human knowledge existing in the form of input values, assertions, event sequences, and human written test suites. The mined knowledge is combined with automated crawling and extends the test suite for the uncovered/unchecked portion of the web application. The approach is implemented in TESTILIZER. Experimental results demonstrate that the approach outperforms random test generators and on average generate test suites improved up to 150% in fault detection rate and 30% in code coverage.

The authors in [4] present a novel technique of exploiting static analysis that guides automated test

generation for binary programs. It also prioritized paths that are to be explored. Initial experiments on a suite of benchmarks from real applications demonstrate the effectiveness of the approach.

The authors in [43] present a comprehensive survey on metamorphic testing. In [38] state of the art trends in symbolic execution are discussed.A comprehensive survey of symbolic execution techniques is presented in [5].

The authors in [10] present results on the "Impact Project Focus Area" on the subject of symbolic execution. Classical and as well as modern symbolic execution techniques are presented.

In [11] symbolic execution is presented along with its illustration with example. The paper also discusses various works in symbolic execution. The authors in [33] present a framework called SIG-Droid for testing android apps using automated program analysis to extract app models. SIG-Droid is implemented and evaluated where the results show the effectiveness of the framework.

In [46] dynamic test data generation framework which forms its basis on genetic algorithms is proposed. The framework has a program analyzer and test case generator which intercommunicate to generate test cases. Efficiency is demonstrated by running it on several programs. Empirical results suggest that the proposed framework is better than the current test data generation methods. The authors in [6] an approach for the systematic selection of input test data is presented. The model is based on key characteristics of model transformation. The input domain is captured in a metamodel.

In [42] genetic algorithm based on an automatic test pattern generation technique is proposed. The proposed scheme achieves higher detection coverage over a greater population of Hardware Trojan Horses (HTH) in ISCAS benchmark circuits.

The authors in [24] present a multifaceted project which automates security testing and robustness of android apps in a scalable manner. An android-specific program analysis technique is proposed which generates a huge number of test cases for app fuzzing. Also, a testbed that generates test cases is developed.

B. RQ 2: Are there security-oriented test case generation techniques in the scientific literature?

The above sections show several research efforts that address test case generation from different perspectives. This section summarized the most work that has been done for test case generation where security sounds like the main perspective. Wimmel and Jurjens [55] generated a test sequence for a transactional system based on a formal security model. They used the specification to generate mutants and attack scenarios to guide the security testing efforts. The authors used mutation testing on specifications to extract those interaction sequences that are most likely to find security issues. Martin and Xie [26] presented a framework and tool called CIRG that generates tests based on change impact analysis. Their experimental results showed that tests can be generated to achieve high structural coverage of policies. The investigated technique is aimed at test generation from access control policy specifications written in XACML (OASIS XACML).

Masson, Pierre-Alain, et al. [56] utilized security policy to compute tests that exercise security properties. Model based testing technique is proposed for verifying if the access control policies are correctly implemented. B language is used to develop the functional model and used for the security test generation. They showed how the test purposes can be automatically computed, by modeling the test needs as syntactic transformation rules that transform regular expressions. Li et al. [57] adopted model-based formal testing of security policies to propose a two-stage approach to generate test cases from a security policy specified in Or-BAC rules. It focuses on the generation of test purposes from individual OrBAC rules. The authors first generated test purposes from Or-bac rules then they generated test cases from these test purposes.

Julliand et al. [58] proposed an approach to generating security tests in addition to functional tests by re-using the functional test model together with a new model of security properties defined by a security engineer. No explicit access control model was used. The test purpose is specified in a language that allows the tester to describe which actions to call and which states to reach. Darmaillacq et al. [59] proposed a semi-automatic rule-based method to generate tests of the conformance of a system to a given security policy. The method generates test cases directly from a security policy expressed as a set of security requirements, using two relations: one between predicates appearing in the rules and elementary test cases used to test predicates in the system, and another one between logical operators and test case combinators.

Marback et al. [25] proposed a security testing approach that derives test cases from the design level. The approach has four activities 1) Build threat trees from threat modeling, 2) Generating security tests from threat, 3) Trees generate test inputs contains valid and invalid inputs, and 4) Assigning input values to parameters. The approach uses manually built threat trees on the grounds of manually constructed data flow diagrams for an application. Abbassi et al. [60] proposed a framework to specify a security policy and to test its implementation. This framework is characterized as follows: (1) the security policy enforcement is specified through a new modeling language, S-Promela, (2) the test criteria are expressed by the use of a temporal logic LTL and (3) the test cases are generated by a classical model checking technique.

Dadeau et al. [61] proposed a mutation based test generation and assessment technique to check the implementation of a security-protocol that is developed in the high level security language. Their approach produced mutant operators that create bug in the security protocols and generate test cases for HLPSL models. Xu et al. [52] proposed an approach to automated generation of security tests formal threat models in the form of predicate/ transition nets. The proposed technique is conducted on two open-source real systems. Their approach successfully discover several security risks, and was showing the capability of killing the injected security mutants injected.

Huang et al. [62] try to combine dynamic and static analysis in order to find security weaknesses in Java based applications. They presented an analyzer to achieve no false negatives and reduce false positives. Bozic et al. [63] used IPO-family algorithm for web security testing. The authors addressed IPOG-F and IPOG algorithms as they are freely available in the ACTS tool. These algorithms produce test cases for investigating and identifying injection attacks that can introduce security breaches. The work addresses cross-site scripting (XSS) vulnerabilities detection.

C. RQ 3: What are the limitation and the opportunities provided by the selected studies?

We discuss our observations and findings in this section. After having reviewed the literature we identified the research challenges, which we discuss in the subsequent paragraphs.

The techniques found in the literature vary according to the techniques and approaches employed to generate test cases. There is a need to study the suitability and feasibility of existing software security test case generation techniques according to the type of software applications. Also, a relationship between the existing techniques and application types needs to be investigated.

Automation is often enabled by AI-based techniques and algorithms. There is a need to investigate existing AI-based techniques and look into their application in software security test case generation algorithms. Also, existing AI-based techniques used in automated software testing can be investigated and enhanced to propose software security test case generation approaches and algorithms.

Software security in terms of test case generation as a research area shows many research limitations and challenges that need to be further addressed and explored. Although we found many opportunities during conducting this review still the challenges might motivate scholars in this field.

To conclude the findings of conducting this review, Table 4 summarizes the opportunities and limitations of the techniques, tools, algorithms found and studied in this systematic review.

Table 4. Summary of Opportunities and Limitations of the Studied Techniques

Technique / Algorithm /Tool	Opportunities	Limitations
[52] Structural Based Software Test Case Generation Technique	Automation of security testing process by generating security test cases by using formal threat models	The cost of generated attack paths is not analyzed.
[15] Functional Based Software Test Case Generation Technique	The proposed technique is a combination of a systematic test case and typestate mining that automates the validation of software. The proposed method incorporates the power of both systematic test cases and typestate mining.	Combining systematic test cases and typestate mining would require a tradeoff to achieve full potential which is not discussed in the paper.
[51] Functional Based Software Test Case Generation Technique	The proposed technique incorporates the power of use case specification and domain modeling.	The behavioral model is considered expensive. Cost analysis of combining the two techniques is not carried out by the

		authors.
[27] Functional Based Software Test Case Generation Technique	The technique for automating web service testing is proposed.	Heterogeneous scenarios of web service i.e. running in the cloud, internet, and web servers are not considered and analyzed in the paper.
[7] Functional Based Software Test Case Generation Technique	A case study of generating test cases in various heterogeneous scenarios is presented.	The approach can be comprehensively validated by considering a heterogeneous industrial real-life application. Where in the paper only one real-life application is considered for validation.
[54] Functional Based Software Test Case Generation Technique	The authors extend metamorphic testing into a user-oriented approach for software verification.	The study can be improved by considering more search engines at the moment only four search engines are analyzed.
[13] Functional Based Software Test Case Generation Technique	Existing research in metamorphic testing and its challenges are discussed.	The authors suggest improvement in metamorphic testing. Suggested improvements can be improved by applying them in various heterogeneous real-life application environments.
[37] Functional Based Software Test Case Generation Technique	A path selection strategy for selecting test cases that kill mutants is proposed.	Cost analysis of the path selection strategy is not analyzed by the authors.
[45] Gray Box Based Software Test Case Generation Technique	The authors present a survey of UML-based testing techniques.	Test case generation from behavioral specification diagrams: sequence, statechart, and activity diagram were considered. Their analysis is not discussed in the paper therefore which one to choose in what kind of scenario needs to be analyzed.
[21] Non-Functional Based Software Test Case Generation Technique	An automated security validation system AppInspector is proposed.	The classification of apps analyzed is not discussed in the paper.
[26] Non-Functional Based Software Test Case Generation Technique	A novel framework for automated test generation for access control policies is put forth along with that supporting tool known as CIRG (Change Impact Request Generation) is used to generate tests based on change impact analysis.	Cost analysis of the proposed CIRG tool is presented in the paper.
[41] Non-Functional Based Software Test Case Generation Technique	Input generation techniques for apps that run in the android operating system are compared.	Evaluation of input generation techniques along with their suitability in respective application scenarios is not analyzed and discussed in the paper.
[20] Random Based Software Test Case Generation Technique	The effectiveness of automated test generation techniques that produce coverage providing tests are analyzed.	Fault detection ability is analyzed but not quantified by the authors in the paper. The work can be improved by quantifying fault detection ability.
[19] Search-Based Software Test Case Generation Technique	The process of identifying bugs and errors is automated with an artificial bee colony technique.	Although the technique is critically reviewed it will be good if the technique is evaluated with heterogeneous scenarios.
[44] Search-Based Software Test Case Generation Technique	A novel approach for generating test paths is put forth	The proposed technique generates a test path where the cost efficiency of test paths is not considered.
[18] Search-Based Software Test Case Generation Technique	A research prototype named EvoSuite automatically generates test suites for Java programming language is written, classes.	The prototype needs to be analyzed over real-life Java-based applications.
[28] Search-Based Software Test Case Generation Technique	A survey of metaheuristic search techniques for automating test data generation for functional and structural testing is presented.	The classification of techniques is not discussed by the authors.

[1] Search-Based Software Test Case Generation Technique	Model-based testing in an industrial setting is automated. Search based OCL constraint solver is proposed.	Cost analysis of the search-based OCL constraint solver is not discussed.
[29] Search-Based Software Test Case Generation Technique	The input generation technique is proposed where the URL is downloaded and split into a token.	The proposed approach is not compared and analyzed against existing input generation techniques.
[25] Search-Based Software Test Case Generation Technique	Security test case derivation from design level artifacts is proposed.	The suitability of design-level artifacts according to application scenarios is not analyzed.
[17] Search-Based Software Test Case Generation Technique	A novel paradigm is proposed where whole test suites are evolved to cover all coverage goals.	Cost analysis of the proposed approach is not performed by the authors.
[2] Data mining Based Software Test Case Generation Technique	The algorithm based on concolic testing is proposed which generates test cases.	Cost analysis of the algorithm is not performed by the authors.
[35] Data mining Based Software Test Case Generation Technique	A novel method of generating test cases based on a black-box autonomous system is proposed. The algorithm involves intelligent searching of state space.	Evaluation and Cost analysis of the searching algorithm is not performed.
[32] Data mining Based Software Test Case Generation Technique	A framework for generating test cases for Javascript-based applications is introduced.	The proposed framework is only applicable to the Javascript-based application. Moreover, the working of the proposed framework should be compared and analyzed with the existing test automation framework. Although some analysis is presented the functionality of the framework is not analyzed.
[22] Data mining Based Software Test Case Generation Technique	A two-phase technique for discovering event sequences that reach a given target line in application code.	Cost evaluation and analysis of the discovery process are not presented.
[8] Data mining Based Software Test Case Generation Technique	A novel technique for alleviating traversed code paths by discarding the ones having side effects similar to a previously explored path.	Evaluation and cost analysis of the technique is not presented and discussed.
[39] Data Mining Based Software Test Case Generation Technique	The white box framework for testing a deep learning framework is put forth.	Empirical evaluation and comparison of the framework with existing frameworks are not presented.
[12] Data Mining Based Software Test Case Generation Technique	The state of the art in dynamic symbolic execution is compiled by the authors.	Classification and comparison of existing symbolic execution techniques are not presented.
[14] Data Mining Based Software Test Case Generation Technique	A customizable and flexible dynamic tainting technique is proposed.	The proposed framework is not compared with existing tainting techniques.
[34] Data Mining Based Software Test Case Generation Technique	A novel method of discovering integer bugs is put forth.	Cost evaluation and analysis of the discovery process are not presented.
[9] Data Mining Based Software Test Case Generation Technique	The technique is proposed which automatically generates test cases by utilizing the code.	Cost evaluation and analysis of discovering bugs are not discussed.
[30] Data Mining Based Software Test Case Generation Technique	A genetic algorithm for the automated generation of test cases is put forth.	A comparison and evaluation of the proposed technique with counterpart algorithms is not presented.
[50] Data Mining Based Software Test Case Generation Technique	Scope bounded testing which develops novel specifications for effectively generating tests for products in the software products line.	The cost evaluation of the proposed technique is not presented.
[31] Data Mining Based Software Test Case Generation Technique	An approach for testing a web application that combines a web crawler and SELENIUM test cases is put forth.	Analysis of mining knowledge is not presented.
[4] Data Mining Based Software Test Case	A technique for automated test generation for binary	Cost evaluation of the proposed techniques is not discussed.

Generation Technique	programs by exploiting static analysis is presented.	
[43] Data Mining Based Software Test Case Generation Technique	A comprehensive survey on metamorphic testing is presented.	Comparison and classification are not presented.
[38] Data Mining Based Software Test Case Generation Technique	A state of the art in symbolic execution is put forth.	The classification of symbolic execution techniques is not presented.
[5] Data Mining Based Software Test Case Generation Technique	A comprehensive survey of symbolic execution techniques is presented.	Evaluation and comparison of techniques are not presented.
[10] Data Mining Based Software Test Case Generation Technique	Results from a real-life project of symbolic execution are put forth.	Classification and comparison of techniques are not discussed.
[11] Data Mining Based Software Test Case Generation Technique	Various efforts in symbolic execution are presented.	A comparison of the techniques is not discussed.
[33] Data Mining Based Software Test Case Generation Technique	Automated android apps testing framework is proposed.	Evaluation of the proposed framework against the existing android testing framework is not discussed.
[46] Data Mining Based Software Test Case Generation Technique	A genetic algorithm-based dynamic test data generation technique is proposed.	Cost evaluation of the framework is not presented.
[6] Data Mining Based Software Test Case Generation Technique	An approach for intelligently selecting the input test data is presented.	Cost evaluation and analysis of proposed techniques are not presented.
[42] Data Mining Based Software Test Case Generation Technique	An automatic test pattern generation technique based on a genetic algorithm is proposed.	A comparison with existing techniques is not discussed. Evaluation of the proposed algorithm is also lacking.
[24] Data Mining Based Software Test Case Generation Technique	Automated security testing of android apps is proposed.	Analysis and evaluation of the proposed testing technique are lacking.

IV. Conclusion

In this paper, we have comprehensively reviewed existing work in software security test case generation approaches and classified them. Existing approaches and techniques are classified into two major categories: Test Type-based software test case generation techniques and Type of algorithms used for software test case generation. Further Test type-based software test case generation techniques are subdivided into four types: Structural based, Functional based, Gray box based, Non-functional based techniques. Also, the category type of algorithms used for software test case generation techniques is subdivided into Random based, Search-based, and Data mining-based techniques. We discuss related work under the discussed classification. In the end, we present our observations and findings based on our review of the literature. We discuss future research challenges.

References

[1] Shaukat Ali, Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel Briand. A search-based ocl constraint solver for model-based test data generation. In 2011 11th International Conference on Quality Software, pages 41–50. IEEE, 2011.

[2] Saswat Anand, Mayur Naik, Mary Jean Harrold, and Hongseok Yang. Automated concolic testing of smartphone apps. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pages 1–11, 2012.

[3] Abdulbaki Aydin, Muath Alkhalaf, and Tevfik Bultan. Automated test generation from vulnerability signatures. Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014, pages 193–202, 03 2014.

[4] Domagoj Babic´, Lorenzo Martignoni, Stephen McCamant, and Dawn Song. Statically-directed dynamic automated test generation. In Proceedings of the 2011 International Symposium on Software Testing and Analysis, pages 12–22, 2011.

[5] Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR), 51(3):1–39, 2018.

[6] Benoit Baudry. Testing model transformations: a case for test generation from input domain models. Model-Driven Engineering for Distributed Real-Time Systems, pages 43–72, 2013.

[7] Eddy Bernard, Bruno Legeard, Xavier Luck, and Fabien Peureux. Generation of test sequences from formal specifications: Gsm 11-11 standard case study. Softw. Pract. Exper., 34(10):915–948, August 2004.

[8] Peter Boonstoppel, Cristian Cadar, and Dawson Engler. Rwset: Attacking path explosion in constraint-based test generation. In International conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 351–366. Springer, 2008.

[9] Cristian Cadar and Dawson Engler. Execution generated test cases: How to make systems code crash itself. In International SPIN Workshop on Model Checking of Software, pages 2–23. Springer, 2005.

[10] Cristian Cadar, Patrice Godefroid, Sarfraz Khurshid, Corina S Pasareanu, Koushik Sen, Nikolai Tillmann, and Willem Visser. Symbolic execution for software testing in practice: preliminary assessment. In 2011 33rd International Conference on Software Engineering (ICSE), pages 1066–1071. IEEE, 2011.

[11] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: three decades later. Communications of the ACM, 56(2):82–90, 2013.

[12] Ting Chen, Xiao-song Zhang, Shi-ze Guo, Hong-yuan Li, and Yue Wu. State of the art: Dynamic symbolic execution for automated test generation. Future Generation Computer Systems, 29(7):1758–1773, 2013.

[13] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. ACM Comput. Surv., 51(1), January 2018.

[14] James Clause, Wanchun Li, and Alessandro Orso. Dytan: a generic dynamic taint analysis framework. In Proceedings of the 2007 international symposium on Software testing and analysis, pages 196–206, 2007.

[15] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller. Automatically generating test cases for specification mining. IEEE Transactions on Software Engineering, 38(2):243–257, 2012.

[16] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. Annalen der Physik, 322(10):891–921, 1905.

[17] Gordon Fraser and Andrea Arcuri. Whole test suite generation. IEEE Transactions on Software Engineering, 39(2):276–291, 2012.

[18] Gordon Fraser and Andrea Arcuri. Evosuite: On the challenges of test case generation in the real world. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pages 362–369. IEEE, 2013.

[19] Disha Garg and Abhishek Singhal. A critical review of artificial bee colony optimizing technique in software testing. In 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), pages 240–244. IEEE, 2016.

[20] Gregory Gay, Matt Staats, Michael Whalen, and Mats PE Heimdahl. The risks of coverage- directed test case generation. IEEE Transactions on Software Engineering, 41(8):803–819, 2015.

[21] Peter Gilbert, Byung-Gon Chun, Landon Cox, and Jaeyeon Jung. Vision: Automated security validation of mobile apps at app markets. Proceedings of the Second International Workshop on Mobile Cloud Computing and Services, 01 2011.

[22] Casper S Jensen, Mukul R Prasad, and Anders Møller. Automated testing with targeted event sequence generation. In Proceedings of the 2013 International Symposium on Software Testing and Analysis, pages 67–77, 2013.

[23] Mohammad Keyvanpour, Hajar Homayouni, and Hossein Shirazi. Automatic software test case generation: An analytical classification framework. International Journal of Software Engineering and its Applications, 6:1–16, 01 2012.

[24] Riyadh Mahmood, Naeem Esfahani, Thabet Kacem, Nariman Mirzaei, Sam Malek, and Angelos Stavrou. A whitebox approach for automated security testing of android applications on the cloud. In 2012 7th International Workshop on Automation of Software Test (AST), pages 22–28. IEEE, 2012.

[25] Aaron Marback, Hyunsook Do, Ke He, Samuel Kondamarri, and Dianxiang Xu. Security test generation using threat trees. In 2009 ICSE Workshop on automation of software test, pages 62–69. IEEE, 2009.

[26] E. Martin and T. Xie. Automated test generation for access control policies via change-impact analysis. In Third International Workshop on Software Engineering for Secure Systems (SESS'07: ICSE Workshops 2007), pages 5–5, 2007.

[27] Evan Martin, Suranjana Basu, and Tao Xie. Automated testing and response analysis of web services. pages 647–654, 07 2007.

[28] Phil McMinn. Search-based software test data generation: a survey. Software testing, Verification and reliability, 14(2):105–156, 2004.

[29] Phil McMinn, Muzammil Shahbaz, and Mark Stevenson. Search-based test input generation for string data types using the results of web queries. In 2012 IEEE Fifth International Conference on

Software Testing, Verification and Validation, pages 141–150. IEEE, 2012.

[30] Christoph C. Michael, Gary McGraw, and Michael A Schatz. Generating software test data by evolution. IEEE transactions on software engineering, 27(12):1085–1110, 2001.

[31] Amin Milani Fard, Mehdi Mirzaaghaei, and Ali Mesbah. Leveraging existing tests in auto- mated test generation for web applications. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pages 67–78, 2014.

[32] Shabnam Mirshokraie, Ali Mesbah, and Karthik Pattabiraman. Jseft: Automated Javascript unit test generation. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pages 1–10. IEEE, 2015.

[33] Nariman Mirzaei, Hamid Bagheri, Riyadh Mahmood, and Sam Malek. Sig-droid: Automated system input generation for android applications. In 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pages 461–471. IEEE, 2015.

[34] David Molnar, Xue Cong Li, and David A Wagner. Dynamic test generation to find integer bugs in x86 binary linux programs. In USENIX Security Symposium, volume 9, pages 67–82, 2009.

[35] Galen E Mullins, Paul G Stankiewicz, and Satyandra K Gupta. Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles. In 2017 IEEE international conference on robotics and automation (ICRA), pages 1443–1450. IEEE, 2017.

[36] Jeff Offutt and Wuzhi Xu. Generating test cases for web services using data perturbation.
ACM SIGSOFT Software Engineering Notes, 29(5):1–10, 2004.

[37] Mike Papadakis and Nicos Malevris. Mutation based test case generation via a path selection strategy. Inf. Softw. Technol., 54(9):915–932, September 2012.

[38] Corina S Pa˘sa˘reanu and Willem Visser. A survey of new trends in symbolic execution for software testing and analysis. International journal on software tools for technology transfer, 11(4):339, 2009.

[39] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In proceedings of the 26th Symposium on Operating Systems Principles, pages 1–18, 2017.

[40] M. Naga Prasanna, S. N. Sivanandam, Rajesh Venkatesan, and R. Sundarrajan. A survey on automatic test case generation. 2011.

[41] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated test input generation for android: Are we there yet? (e). pages 429–440, 11 2015.

[42] Sayandeep Saha, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, Debdeep Mukhopadhyay, et al. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In International Workshop on Cryptographic Hardware and Embedded Systems, pages 577–596. Springer, 2015.

[43] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. A survey on metamorphic testing. IEEE Transactions on software engineering, 42(9):805–824, 2016.

[44] AVK Shanthi, G MohanKumar, et al. A novel approach for automated test path generation using tabu search algorithm. International Journal of Computer Applications, 48(13):28–34, 2012.

[45] Mahesh Shirole and Rajeev Kumar. Uml behavioral model based test case generation: A survey. SIGSOFT Softw. Eng. Notes, 38(4):1–13, July 2013.

[46] Anastasis A Sofokleous and Andreas S Andreou. Automatic, evolutionary test data generation for dynamic software testing. Journal of Systems and Software, 81(11):1883–1898, 2008.

[47] Hitesh Tahbildar and Bichitra Kalita. Automated software test data generation: direction of research. International Journal of Computer Science and Engineering Survey, 2(1):99–120, 2011.

[48] H. H. Thompson. Why security testing is hard. IEEE Security Privacy, 1(4):83–86, 2003.

[49] Herbert H Thompson and James A Whittaker. Testing for software security. Dr. Dobb's Journal: Software Tools for the Professional Programmer, 27(11):24–28, 2002.

[50] Engin Uzuncaova, Sarfraz Khurshid, and Don Batory. Incremental test generation for software product lines. IEEE transactions on software engineering, 36(3):309–322, 2010.

[51] Chunhui Wang, Fabrizio Pastore, Arda Goknil, Lionel Briand, and Zohaib Iqbal. Automatic generation of system test cases from use case specifications. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, page 385–396, New York, NY, USA, 2015. Association for Computing Machinery.

[52] D. Xu, M. Tu, M. Sanford, L. Thomas, D. Woodraska, and W. Xu. Automated security test generation with formal threat models. IEEE Transactions on Dependable and Secure Computing, 9(4):526–540,

2012.

[53] Dianxiang Xu, Manghui Tu, Michael Sanford, Lijo Thomas, Daniel Woodraska, and Weifeng Xu. Automated security test generation with formal threat models. IEEE transactions on dependable and secure computing, 9(4):526–540, 2012.

[54] Z. Q. Zhou, S. Xiang, and T. Y. Chen. Metamorphic testing for software quality assessment: A study of search engines. IEEE Transactions on Software Engineering, 42(3):264–284, 2016.

[55] Wimmel, Guido, and Jan Jürjens. "Specification-based test generation for security-critical systems using mutations." In International Conference on Formal Engineering Methods, pp. 471-482. Springer, Berlin, Heidelberg, 2002.

[56] Masson, Pierre-Alain, Jacques Julliand, Jean-Chritophe Plessis, Eddie Jaffuel, and Georges Debois. "Automatic generation of model based tests for a class of security properties." In Proceedings of the 3rd international workshop on Advances in model-based testing, pp. 12-22. 2007.

[57] Li, Keqin, Laurent Mounier, and Roland Groz. "Test generation from security policies specified in or-bac." In 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 2, pp. 255-260. IEEE, 2007.

[58]] Julliand, Jacques, Pierre-Alain Masson, and Regis Tissot. "Generating security tests in addition to functional tests." In Proceedings of the 3rd international workshop on Automation of software test, pp. 41-44. 2008.

[59] Darmaillacq, Vianney, Jean-Luc Richier, and Roland Groz. "Test generation and execution for security rules in temporal logic." In 2008 IEEE International Conference on Software Testing Verification and Validation Workshop, pp. 252-259. IEEE, 2008.

[60] Abbassi, Ryma, and Sihem Guemara El Fatmi. "Towards a test cases generation method for security policies." In 2009 International Conference on Telecommunications, pp. 41-46. IEEE, 2009.

[61] Dadeau, Frédéric, Pierre-Cyrille Héam, and Rafik Kheddam. "Mutation-based test generation from security protocols in HLPSL." In 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, pp. 240-248. IEEE, 2011.

[62] Huang, Yu-Yu, Kung Chen, and Shang-Lung Chiang. "Finding security vulnerabilities in Java Web applications with test generation and dynamic taint analysis." In Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, pp. 133-138. Springer, Berlin, Heidelberg, 2012.

[63] Bozic, Josip, Bernhard Garn, Dimitris E. Simos, and Franz Wotawa. "Evaluation of the IPO-family algorithms for test case generation in web security testing." In 2015 IEEE Eighth International Conference on Software Testing, Verification and Valid

Dr. Mamdouh Alenezi is currently the Dean of Educational Services at Prince Sultan University. Dr. Alenezi received his MS and Ph.D. degrees from DePaul University and North Dakota State University in 2011 and 2014, respectively. He has extensive experience in data mining and machine learning where he applied several data mining techniques to solve several Software Engineering problems. He conducted several research areas and development of predictive models using machine learning to predict fault-prone classes, comprehend source code, and predict the appropriate developer to be assigned to a new bug.

Dr. Mohammed Akour is an associate Professor in Software Engineering at Yarmouk University (YU). He got his Bachelor (2006) and Master (2008) degree from Yarmouk University in Computer Information Systems with Honor. He joined Yarmouk University as a Lecturer in August 2008 after graduating with his master in Computer Information Systems. He joined Yarmouk University again in April 2013 after graduating with his PhD in Software Engineering from NDSU with Honor. He serves as Key Note Speaker, Organizer, a Co-chair and publicity Chair for several IEEE conferences, and as ERB for more than 10 ISI indexed prestigious journals. He is a member of the International Association of Engineers (IAENG). Dr. Akour at Yarmouk University served as Head of accreditation and Quality assurance for two years and then was hired in 2017 as director of computer and Information Center. In 2018, Dr. Akour has been hired as vice Dean of Student Affairs at Yarmouk University.