JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
Volume 06 Issue 01
2023

Review Article
https://doi.org/10.56979/601/2023

Enhancing Security Testing Through Evolutionary Techniques: A Novel Model

M. Imran Khan Khalil[1], Afsheen Gul[1], Amer Taj[1], Asif Nawaz[2], Naveed Jan[3], and Sheeraz Ahmad[4*]

[1]University of Engineering & Technology, Peshawar, 25000, Pakistan
[2]Engineering Technology and Science, Higher Collages of Technology, UAE
[3]University of Technology, Nowshera, 24100, Pakistan
[4]Iqra National University, Peshawar, 25000, Pakistan
[*]Corresponding Author: Sheeraz Ahmad. Email: sheeraz.ahmad@inu.edu.pk

Received: August 11, 2023 **Accepted:** November 23, 2023 **Published:** December 05, 2023

Abstract: Software systems are integral to modern organizations, necessitating rigorous testing to ensure security and integrity. However, with the evolution of technology, vulnerabilities and threats to software security are on the rise. Metaheuristic algorithms (MHS) or evolutionary techniques have emerged as valuable tools in addressing these challenges. This research aims to explore and evaluate evolutionary software security testing techniques comprehensively. Specific objectives include analyzing different test cases and strategies, identifying commonly targeted security vulnerabilities, assessing cost-effective and scalable testing techniques, and developing a framework for selecting optimal evolutionary testing methods. The methodology employs a systematic literature review across five major databases, selecting 52 relevant papers. Findings indicate prevalent security vulnerabilities such as Cross-site scripting XSS, Buffer overflow/stack overflow, SQL/XML injection, etc. The commonly used genetic algorithms for software security testing are Genetic algorithm, Particle swarm optimization, and Simulated annealing. Cost-effective and scalable MHS algorithms are ranked, with the Genetic algorithm emerging as the most effective. Additionally, a model for selecting and utilizing MHS algorithms is proposed based on research findings. This study offers valuable insights for researchers and practitioners, outlining future research avenues and providing practical guidelines for employing MHS algorithms in software security testing.

Keywords: Security Testing Techniques, Metaheuristic Algorithms, Software Security Testing, Security Vulnerabilities, Evolutionary Algorithms.

1. Introduction

With the rapid technological advancements, organizations increasingly rely on information systems (IS) and information technology (IT) to drive their operations [1], [2]. This dependence underscores the importance of integrating various software and hardware systems to facilitate digital transformations. Notably, many of the world's most successful companies are entrenched in IT and IS, highlighting the significance of these technologies in today's business landscape [3]. However, the widespread adoption of IT and IS exposes organizations to many security threats. As businesses embrace digital transformations, they must prioritize the security of their IT and IS infrastructure. This involves implementing robust security measures to safeguard against potential vulnerabilities and attacks [4], [5], and [6].

Central to these security efforts is the development of secure software systems. In an era defined by technological innovation, developers face the challenge of ensuring the security of software systems amidst evolving threats. Malicious attacks, such as viruses and hacking attempts, pose significant data integrity and confidentiality risks [7].

Software security testing has emerged as a critical practice to address these challenges. Software security testing focuses on identifying and mitigating vulnerabilities within software systems to ensure their

integrity and functionality. By conducting thorough security testing, organizations can mitigate risks associated with malware, viruses, and denial of service (DoS) attacks [8], [9], and [10].

Software security testing encompasses various techniques, including both conventional and evolutionary approaches [11], [12]. Conventional techniques include formal testing, fuzzy testing, and white-box testing, while evolutionary techniques leverage metaheuristic algorithms to enhance testing effectiveness and efficiency [13].

Despite the effectiveness of evolutionary testing techniques, developers need help selecting the most suitable approach for software security testing. When choosing a testing technique, cost, scalability, and effectiveness must be carefully considered. Moreover, the evolving nature of security threats necessitates ongoing refinement and adaptation of testing methodologies [14].

This research aims to address these challenges by providing recommendations for developers on selecting evolutionary testing techniques for software security testing [15]. This study seeks to identify the most effective and cost-efficient evolutionary testing techniques for different security vulnerabilities and fault types by conducting a systematic literature review and analysing existing research. Through developing a comprehensive framework, this research aims to empower developers with the tools and insights needed to navigate the complexities of software security testing effectively [16] and [17].

2. Literature Review and Background Study

This section presents an in-depth review of the literature on software security testing, emphasizing its significance and various conventional and evolutionary testing techniques. Additionally, it outlines the systematic literature review (SLR) process employed in this study.

2.1 Software Security Testing

Software security testing is crucial for assessing the ability of installed programs to protect data and ensure the functionality of business information systems by mitigating vulnerabilities and threats [18]. With businesses relying on software systems to manage operations and store vital information, ensuring robust security measures is imperative in today's digital landscape. Once considered secure, the internet now poses significant risks, making it essential for organizations to prioritize data security and prevent theft and online scams [19].

Software security testing is a layered approach, identifying gaps and selecting appropriate security measures tailored to specific business requirements [20]. It aids in error identification, threat detection, risk analysis, and proactive design to mitigate potential security breaches. Security testing techniques play a vital role in safeguarding against common vulnerabilities such as SQL injection and arbitrary file uploads [21].

Several software security testing methods exist, including formal testing, model-based testing, fault injection-based testing, fuzzy testing, vulnerability scanning, property-based testing, and white-box testing [22]. These techniques offer diverse approaches to identifying and addressing security vulnerabilities, each with advantages and limitations.

2.2 Software Security Testing Techniques

Software security testing techniques encompass both conventional and evolutionary approaches. Conventional techniques, such as formal and fault injection-based testing, focus on structured methodologies for identifying vulnerabilities [23], [24], and [25]. These methods provide valuable insights but may need more scalability and effectiveness.

Evolutionary testing techniques, such as genetic algorithms, taint analysis, dynamic symbolic execution, and metamorphic testing, offer innovative solutions to emerging security challenges [26, 27-31]. These techniques leverage heuristic search algorithms and adaptive strategies to assess and improve software security dynamically.

2.3 Implications of Evolutionary Testing Techniques for Software Security Testing

Evolutionary testing techniques demonstrate adaptability and effectiveness in addressing complex

security concerns. Genetic algorithms optimize solutions for various problems, including NP-hard and NPC, by iteratively evaluating and refining solutions [32]. Taint analysis tracks variable status to detect vulnerabilities, while dynamic symbolic execution analyses software behavior to identify potential security risks [33-35]. Metamorphic testing ensures software functionality without ideal oracles, enhancing security assurance [36].

2.4 Systematic Literature Review

A systematic literature review (SLR) methodology was employed to synthesize existing research on software security testing. This method provides evidence-based insights, identifies research gaps, and guides new research directions [37] and [38].

The SLR process involves planning, conducting, and reporting phases [39]. Research questions are formulated during planning, and a review protocol is developed. The conducting phase involves identifying relevant studies, defining search criteria, and analyzing extracted data. Finally, the reporting phase concludes, addresses threats, and presents research findings.

2.5 Guiding Framework

A guiding framework was developed to assist in selecting evolutionary testing techniques for software security testing. This framework considers test case and level, test strategy, security vulnerabilities and fault types, effectiveness, cost-friendliness, and scalability as criteria for recommending suitable techniques [40]. The framework aims to guide developers in selecting evolutionary testing techniques based on specific testing requirements and constraints.

3. Research Methodology

This section outlines the methodology employed in this research, focusing on a systematic literature review approach to address the research questions and gain insights into the field. The systematic literature review allows for identifying, selecting, and critically analyzing relevant literature and assessing its quality, validity, and limitations [41]. Below, the stages of the research methodology are elucidated:

3.1 Reviewing Planning

3.1.1 Need for Systematic Review

Given the significance of software security testing in ensuring the integrity of software systems, particularly concerning metaheuristic algorithms and evolutionary testing techniques [42], a systematic literature review is warranted. This review aims to identify the evolutionary testing techniques used in various scenarios for software security testing, culminating in developing a model to recommend specific techniques based on requirements.

3.1.2 Research Questions Specification

Defining straightforward research questions is pivotal for guiding the search strategy and data extraction process [43]. The following research questions (RQ) have been formulated for this study:

- RQ1: What constitutes the research space for evolutionary testing of software security? This question delves into contemporary research in the field, encompassing aspects such as security vulnerabilities, metaheuristic algorithms, and test strategies.
- RQ2: Which evolutionary testing techniques are cost-friendly, effective, and scalable for software security testing across different vulnerabilities? This question evaluates the cost-effectiveness, efficacy, and scalability of various evolutionary testing techniques for software vulnerabilities.

3.1.3 Development of Review Protocol

The review protocol delineates the methods, criteria, and study selection strategy employed in the systematic literature review.

- Study Selection Strategy: The selection strategy involves using predefined keywords to query specific databases and repositories. The selected databases include ACM Digital Library, IEEE Xplore, Science Direct, Springer, and Wiley Interscience, along with leading software engineering journals such as IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodologies.
- Inclusion and Exclusion Criteria: Papers meeting inclusion criteria (e.g., relevance to research questions, peer-reviewed status, publication between 2010 and 2021) were selected, while those failing to meet these criteria were excluded.
- Quality Assessment: The selected papers were evaluated based on ten criteria to ascertain their quality and relevance to the research.
- Data Extraction, Synthesis, and Analysis: Relevant information about the research questions was extracted from the selected papers and synthesized for analysis. This process involved two steps: general data extraction and extraction of information directly related to the research questions.

This systematic literature review aims to provide comprehensive insights into evolutionary testing techniques for software security, addressing the defined research questions and contributing to the existing body of knowledge in the field.

3.2 Threats to Validity

The study's credibility is influenced by threats to validity, which can adversely affect its outcome. To address these concerns, several principles should be taken into account. Firstly, construct validity ensures that the measurement methods align with the studied concepts, ensuring accurate data collection. Qualitative research involves providing clear explanations and understanding interview questions to avoid confusion. Additionally, serving as an external reviewer helps verify the research proposal, minimizing subjective biases. Although subjectivity may exist, gathering ample information and using diverse respondent samples help mitigate this risk.

Internal validity, which assesses causal relationships and researcher control over variables are not a focus of this study on evolutionary software security techniques. While internal validity concerns may arise from biases during analysis and coding, this study does not aim for statistical causal relationships in software development practices. Conclusion validity examines researcher bias during data interpretation, which is challenging to eliminate. To mitigate this bias, two authors independently analyzed primary papers, maintaining comprehensive records of all 52 papers studied. This dual-author approach was also employed during data collection, analysis, and reporting to ensure consistency and reduce bias. External validity, about the generalizability of study results, is addressed by following established principles. Despite the inherent challenge of eliminating investigator bias, involving two authors in primary paper analysis and maintaining detailed records of retrieved articles help mitigate this bias. This rigorous approach extends to data collection, analysis, and reporting, enhancing the study's validity and reliability.

4. Results

This section presents the findings from analyzing 52 papers from five databases. It overviews these studies, including the publication year, methodology, article type, and publisher. Additionally, Section 5.2 outlines the assessment of the selected papers' quality for this systematic literature review.

The figure below is the number of primary studies included in the systematic literature review:

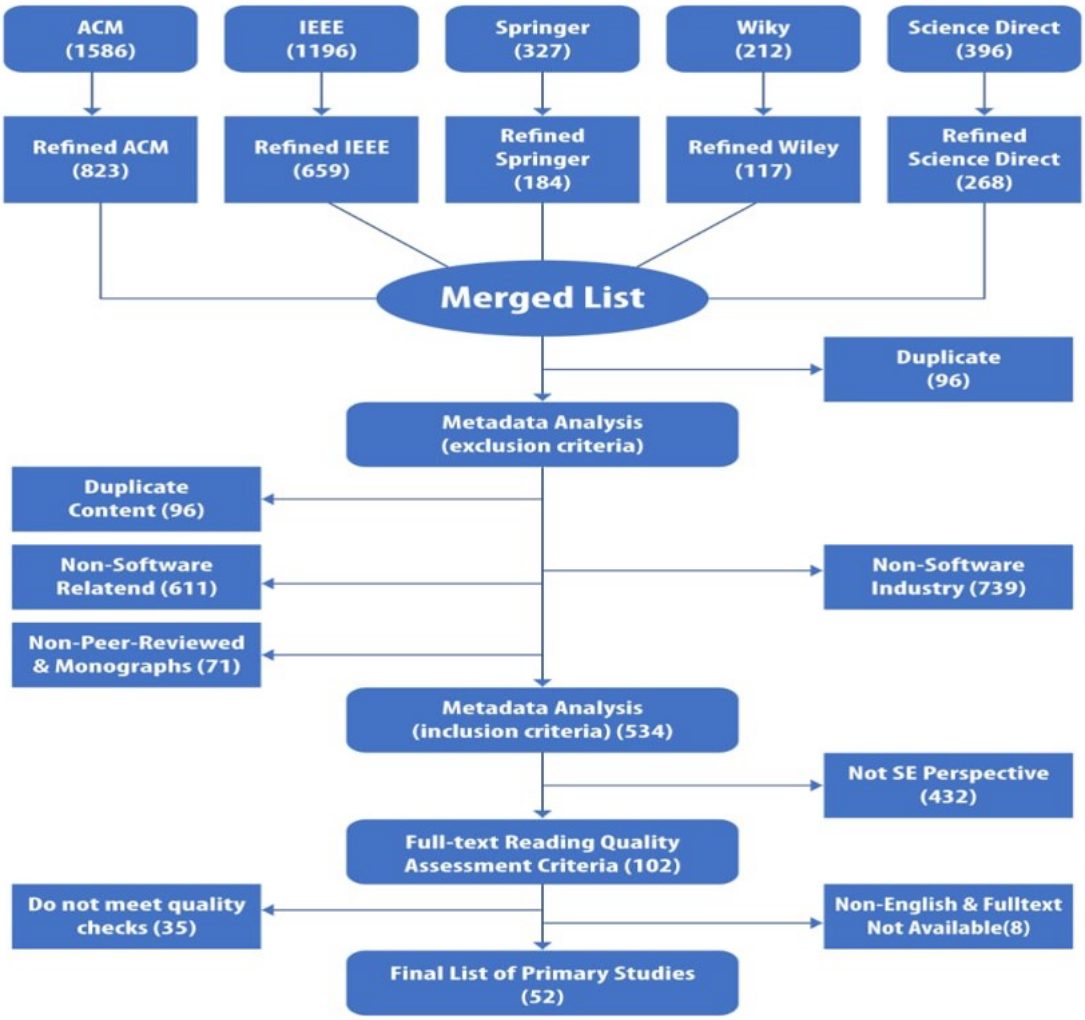

Figure 1. Primary Studies Included in SLR

4.1 Overview of Primary Studies

The data reveals that 48% of the studies were published before 2015, while 52% were published in 2015 or later, indicating a recent trend towards software security techniques. These studies employ various methods and techniques, primarily focusing on software security development activities. For example, [65] introduces a method for ensuring software security in program creation, while [74] analyzes and evaluates security features in software requirements, emphasizing the importance of such techniques in software development.

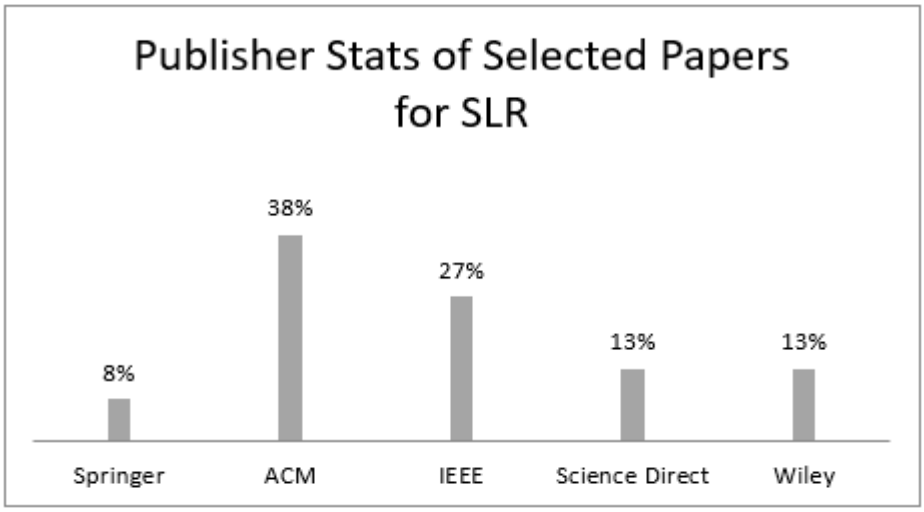

Figure 2. Publisher Statistics of Selected Papers of SLR

The statistics indicate that the majority of articles were chosen from ACM, accounting for 38%, followed by IEEE with 27%. Wiley and Science Direct each contributed 13% of the selected articles, while Springer accounted for only 8% of the final selected list.

4.2 Evaluation of SLRs Quality

The quality of the research papers is assessed based on the evaluation of ten criteria outlined in the table below [51].

Table 1. Evaluation of SLRs Quality

Primary Studies	Research	Aim	Context	Design	Sampling	Control	Data Collection	Reflexivity	Finding	Value	Total
[42]	1	1	1	1	1	1	1	1	1	1	10
[43]	1	1	1	1	1	1	1	1	1	1	10
[44]	1	0	1	0	1	1	0	0	1	1	6
[45]	1	1	1	1	1	1	1	1	1	1	10
[46]	1	1	1	0	0	1	1	0	1	0	6
[47]	1	1	1	0	1	0	1	0	1	1	7
[48]	1	0	1	1	1	0	1	1	0	1	7
[49]	1	0	0	1	1	1	1	1	1	1	8
[50]	1	1	0	0	0	1	1	1	1	1	7
[51]	1	1	1	1	1	1	1	1	1	1	10
[52]	1	1	0	0	1	0	1	1	1	1	7
[53]	1	1	1	1	1	1	1	1	1	1	10
[54]	1	1	1	1	1	1	1	1	1	1	10
[55]	1	1	0	0	0	0	1	1	1	1	6
[56]	1	1	1	1	1	0	0	0	1	1	7
[57]	1	1	1	1	0	0	0	0	1	1	6
[58]	1	0	1	0	1	0	0	1	1	1	6
[59]	1	1	1	1	1	1	1	1	1	1	10
[60]	1	1	1	1	1	1	1	1	1	1	10
[61]	1	1	0	0	0	0	0	1	1	1	5
[62]	1	1	1	1	0	0	0	0	0	1	5
[63]	1	1	1	1	1	1	1	1	1	1	10
[64]	1	1	1	1	1	1	0	0	0	1	7
[65]	1	0	1	1	1	0	1	1	0	1	7
[66]	1	0	0	1	1	1	1	1	1	1	8
[67]	1	1	0	0	0	1	1	1	1	1	7
[68]	1	1	1	1	1	1	1	1	1	1	10
[69]	1	1	0	0	1	0	1	1	1	1	7
[70]	1	1	1	1	1	1	1	1	1	1	10
[71]	1	1	1	1	1	1	1	1	1	1	10
[72]	1	1	0	0	0	0	1	1	1	1	6
[73]	1	1	1	1	1	0	0	0	1	1	7
[74]	1	1	1	1	0	0	0	0	1	1	6

[75]	1	0	1	0	1	0	0	1	1	1	6
[76]	1	1	0	0	0	1	1	0	0	1	5
[77]	1	0	1	0	1	1	0	0	1	1	6
[78]	1	1	1	1	1	1	1	1	1	1	10
[79]	1	1	1	0	0	1	1	0	1	0	6
[80]	1	1	1	0	1	0	1	0	1	1	7
[81]	1	0	1	1	1	0	1	1	0	1	7
[82]	1	1	1	1	1	1	1	1	1	1	10
[83]	1	1	0	0	0	1	1	1	1	1	7
[84]	1	1	1	1	1	1	1	1	1	1	10
[85]	1	1	0	0	1	0	1	1	1	1	7
[86]	1	1	1	1	1	1	1	1	1	1	10
[87]	1	1	1	1	1	1	1	1	1	1	10
[88]	1	1	1	1	1	1	1	1	1	1	10
[89]	1	1	1	1	1	0	0	0	1	1	7
[90]	1	1	1	1	1	0	1	0	1	1	8
[91]	1	1	1	1	1	1	1	1	1	1	10
[92]	1	1	1	1	1	1	1	1	1	1	10
[93]	1	1	1	1	1	1	1	1	1	1	10

To gauge the quality of the 52 primary papers, each study underwent assessment using the framework outlined earlier. The resulting quality assessment is depicted in the table provided. A value of either "0" or "1" was assigned for each characteristic. A "1" indicated the presence of the characteristic, while a "0" denoted its absence. The primary studies in the systematic literature review demonstrated a research-oriented approach, each with a clearly defined aim to align with the review's design. Notably, several studies needed to employ sampling techniques. However, these primary studies thoroughly elucidated their data collection and analysis methods, emphasizing their practical significance. Our analysis revealed that approximately 41% (21 out of 52) of the primary studies obtained total points in the quality assessment.

5. Findings and Discussions

This study employs a systematic literature review to analyze 52 primary studies, aiming to comprehend the methodologies and applications of evolutionary software security techniques utilized by developers and commercial organizations. This chapter presents the results and discussions derived from the systematic literature review, which encompassed 52 papers. Revisiting the research questions within this section is crucial for effectively addressing them. The subsequent sections outline the research questions and their corresponding findings obtained through the SLR.

Research Question 1 (RQ1): What is the research space for evolutionary software security testing?

This overarching question is divided into three critical inquiries focusing on the security vulnerabilities observed in software systems, the prevalence of Metaheuristic search algorithms in software testing, and the frequency of usage of evolutionary testing techniques.

RQ 1.1: Which security vulnerabilities are predominantly observed in software systems?

In today's dynamic business landscape, companies strive to develop software security techniques that afford their systems increased protection against security threats, fostering a risk-free environment. Managers seek flexibility in addressing these concerns to integrate software security techniques into their operations effectively.

Robust software security techniques are crucial for safeguarding company decisions and maintaining competitiveness [44]. These techniques are tailored to meet the unique needs and requirements of the

operational contexts in which companies operate. Based on the findings of the SLR conducted, the following table presents the software security vulnerabilities most commonly observed in software systems:

Table 2. Identified security vulnerabilities

Security Vulnerabilities	Reference Papers	Total Papers	Percentage
Buffer Overflow/Stack Overflow	[52],[54], [55], [58], [74], [83], [86], [88], [89], [91], [91], [93]	12	23%
SQL/XML Injection	[52], [54], [57], [75], [81], [83], [86], [89], [90], [91], [91], [93]	12	23%
Cross-site Scripting XSS (Reflected and Stored)	[52], [53], [54], [57], [66], [68], [69], [71], [75], [81], [83], [86], [87], [88], [89], [90], [91], [92]	18	35%
Broken Algorithms	[54], [73], [83], [89], [90], [91], [92]	7	13%
Privileges, Permissions and Access Control	[55], [56], [60], [73], [81], [83], [86], [88], [89], [90], [91], [92]	12	23%
Spoofing	[55], [81], [83], [86], [89], [90], [91], [92]	8	15%
Undesired Feature Interactions/DoS	[58], [60], [73], [86], [90], [91]	6	12%
Configuration Errors	[58], [60], [83]	3	6%
Floating-point inaccuracies	[68]	1	2%
String Termination Errors	[88]	1	2%
Missing data encryption	[54], [81], [93]	3	6%
Cryptographic Issues	[55], [56], [81]	3	6%
Improper Input Validation	[55], [56]	2	4%
Resource Management Errors	[55], [56], [60]	3	6%
Information Exposure	[55]	1	2%
Fuzzing	55, 81	2	4%

The table illustrates that the most prevalent software security vulnerability is cross-site scripting XSS (reflected and stored), occurring in 35% of the papers in the SLR. Following closely are other frequently encountered vulnerabilities, such as buffer overflow, stack overflow, SQL and XML injection, privileges, permission, and access control, collectively observed in 23% of the papers. Additionally, spoofing is documented in 15% of the papers, while undesired feature interactions and Denial of Service (DoS) are identified in 12%. Broken algorithms are noted in 13% of the papers. While these vulnerabilities are commonly observed in software systems, the table highlights other notable vulnerabilities, including fuzzing, information exposure, resource management errors, improper input validation, cryptographic issues, missing data encryption, string termination errors, floating point inaccuracies, and configuration errors. The figure 3, below visually represents the most frequently observed software security vulnerabilities based on the data provided:

RQ 1.2: What metaheuristic search algorithms are employed for the security testing of a system?

Numerous techniques showcased in the primary studies underscore the significance of these methodologies for securing systems, databases, and software systems within companies. Companies make significant investments to ensure the effective implementation of these techniques, which serve as protective measures and benchmarks for maximizing system security. Among the primary studies, considerable attention has been directed towards various metaheuristic search algorithms or evolutionary software security testing techniques.

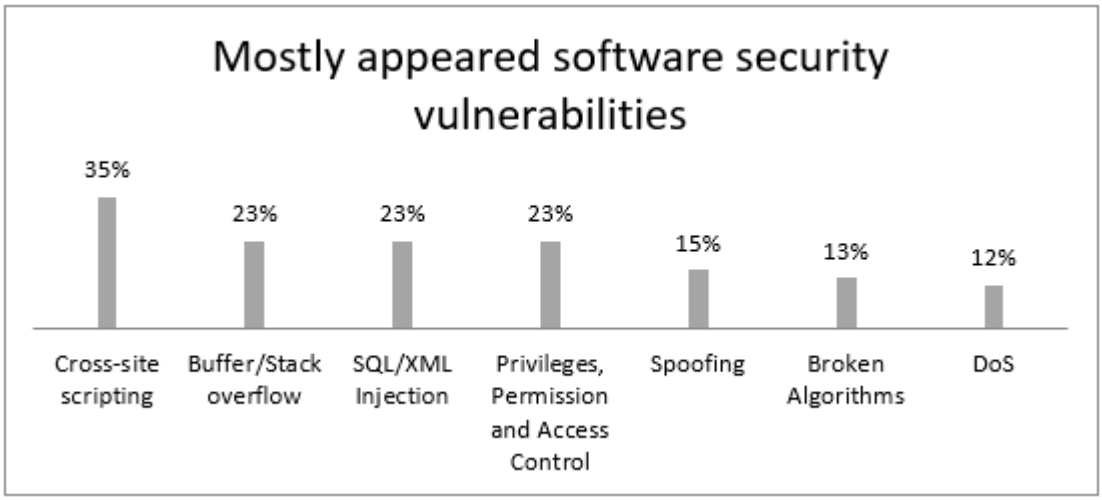

Figure 3. Mostly Appeared Software Security Vulnerabilities

The table indicates the vast array of evolutionary techniques utilized by developers for testing software security vulnerabilities. These encompass hill climbing, greedy algorithms, symbolic execution, differential evolution techniques, bat algorithms, firefly algorithms, taint analysis, static code analysis, combinatorial testing, penetration testing, artificial bee colony algorithms, ant colony optimization, particle swarm optimization, genetic algorithms, evolutionary computation, guided local search, variable neighbourhood search, iterative local search, and simulated annealing. The appearance of these techniques and algorithms across different papers underscores their practical implementation within software organizations.

Table 3. Identified metaheuristic algorithms

Metaheuristic Algorithms	Reference Papers	Total	%
Simulated Annealing	[52], [56], [60], [62], [63], [68], [70], [73], [85], [88], [90], [93], [43], [44]	14	27%
Iterative Local Search	[52]	1	2%
Variable Neighborhood Search	[52]	1	2%
Guided Local Search	[52], [61]	2	4%
Evolutionary Computation	[52]	1	2%
Genetic Algorithm	[52], [60], [61], [62], [65], [66], [68], [69], [71], [74], [78], [82], [84], [85], [90], [91], [92], [93], [42], [43], [44], [47], [46], [45]	24	46%
Particle Swarm Optimization	[52], [59], [62], [65], [69], [70], [84], [85], [88], [90], [93], [43], [44], [45], [46]	15	29%
Ant Colony Optimization	[52], [57], [59], [62], [64], [65], [70], [85], [92]	9	17%
Artificial Bee Colony	[52], [65]	2	4%
Penetration Testing	[53], [58], [79]	3	6%
Combinatorial Testing	[53], [59], [92]	3	6%
Taint Analysis	[57], [60], [66], [69], [76], [79], [82], [88], [92], [93]	10	19%
Static Code Analysis	[64], [69], [74], [75], [82], [88], [90]	7	13%
Firefly Algorithm	[65]	1	2%

Bat Algorithm	[65]	1	2%
Differential Evolution Technique	[67]	1	2%
Symbolic Execution	[70], [71], [72], [76], [77], [80], [83], [85], [87], [90], [93], [46], [47]	12	23%
Greedy Algorithm	[74], [82], [84], [85], [86], [87], [90], [91]	8	15%
Hill Climbing	[85], [87], [92], [66], [67]	5	10%

RQ 1.3: Which evolutionary testing techniques are utilized most frequently?

Studies emphasize that companies operating in dynamic environments highly embrace specific implemented software security testing techniques, such as security testing, security auditing, and risk assessment. These techniques are crucial for fortifying systems or companies, shielding them against potential threats [80]. These techniques must be agile and up-to-date to effectively address vulnerabilities and mitigate risks, thus ensuring the security of companies' software and supporting their competitive and dynamic decision-making processes. Among these implemented and utilized evolutionary techniques, the most frequently employed ones are outlined below:

Table 4. Identified mostly used metaheuristic algorithms

Mostly Used MHS Algorithms/Evolutionary testing Techniques	Total Papers	%
Genetic Algorithm	24	46%
Particle Swarm Optimization	15	29%
Simulated Annealing	14	27%
Symbolic Execution	12	23%
Taint Analysis	10	19%
Ant Colony Optimization	9	17%

The genetic algorithm emerges as the most commonly utilized evolutionary technique, featuring in 46% of the selected papers for this systematic literature review (SLR). Following closely, particle swarm optimization is the second most utilized evolutionary technique, appearing in 29% of the papers. Simulated annealing is reported in 27% of the papers, while symbolic execution is observed in 23%. Additionally, 19% and 17% of the papers found taint analysis and ant colony optimization, respectively. Consequently, these techniques are among the most commonly employed evolutionary/MHS algorithms for software security testing.

The following graph provides a visual representation of the predominant and frequently used evolutionary techniques:

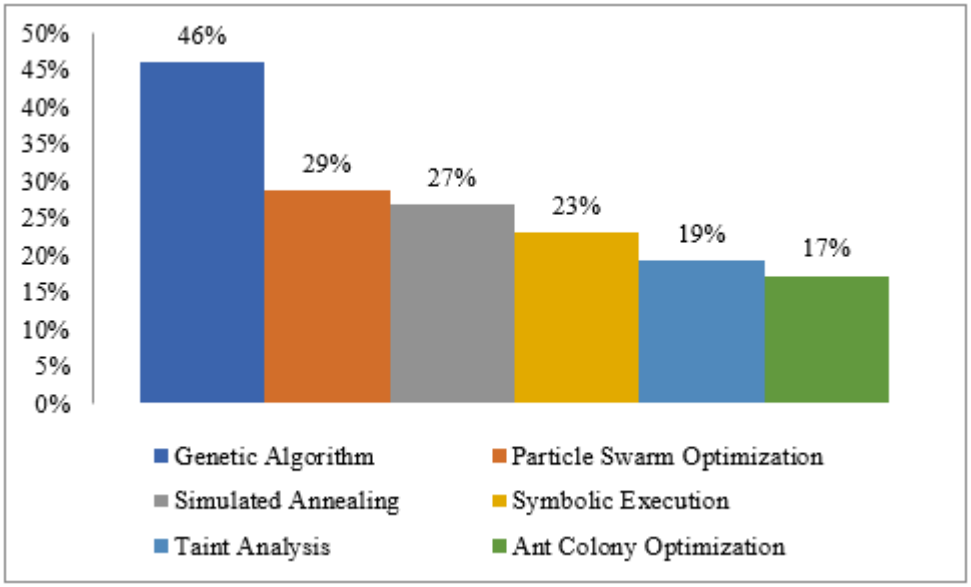

Figure 4. Mostly Used Evolutionary Testing Techniques

Research Question 2 (RQ2): Which evolutionary testing techniques are cost-friendly, effective, and scalable for software security testing across various vulnerabilities?

This question delves into evolutionary testing techniques' cost-friendliness, effectiveness, and scalability for diverse test cases and vulnerabilities. It seeks to explore different evolutionary testing methods that offer cost-effectiveness, efficiency, and adaptability across varied scenarios. RQ2 is further segmented into the following sub-questions:

RQ 2.1: Which evolutionary software security testing technique is more cost-effective in testing software security across different vulnerabilities?

This sub-question specifically targets the cost-effectiveness of various evolutionary security testing techniques. Cost-effectiveness in this context implies affordability for developers and efficiency in achieving desired outcomes. Different evolutionary techniques may exhibit varying levels of cost-effectiveness depending on the specific case or requirement. Thus, assessing these techniques' cost-effectiveness requires examining their applicability and efficiency in different scenarios. For this study, through systematic literature review (SLR), the cost-effectiveness of evolutionary testing techniques is evaluated based on insights gathered from various research articles. These articles explore different techniques for security testing and conclude which ones are most cost-effective. The frequency of appearance of a particular technique in research articles and authors' conclusions regarding its cost-effectiveness serve as an indicator for evaluating its efficacy. Based on the findings, the following evolutionary security testing techniques emerge as the most cost-effective:

Table 5. Identified cost-effective evolutionary testing techniques

Cost Effective Evolutionary Testing Techniques	Reference Papers	Total Papers	%
Simulated Annealing	[59], [52], [60], [63], [88], [78], [67]	7	13%
Particle Swarm Optimization	[59], [62], [65], [84], [88], [93], [90], [46]	8	15%
Genetic Algorithm	[52], [61], [62], [66], [69], [71], [78], [84], [90], [91], [93], [44], [45], [56], [67]	15	29%
Taint Analysis	[57], [69], [76], [79], [82], [88], [92]	7	13%
Ant Colony Optimization	[59], [64], [65], [70], [85], [90]	6	12%
Symbolic Execution	[70], [72], [77], [80], [83], [85], [87], [90], [92]	9	17%

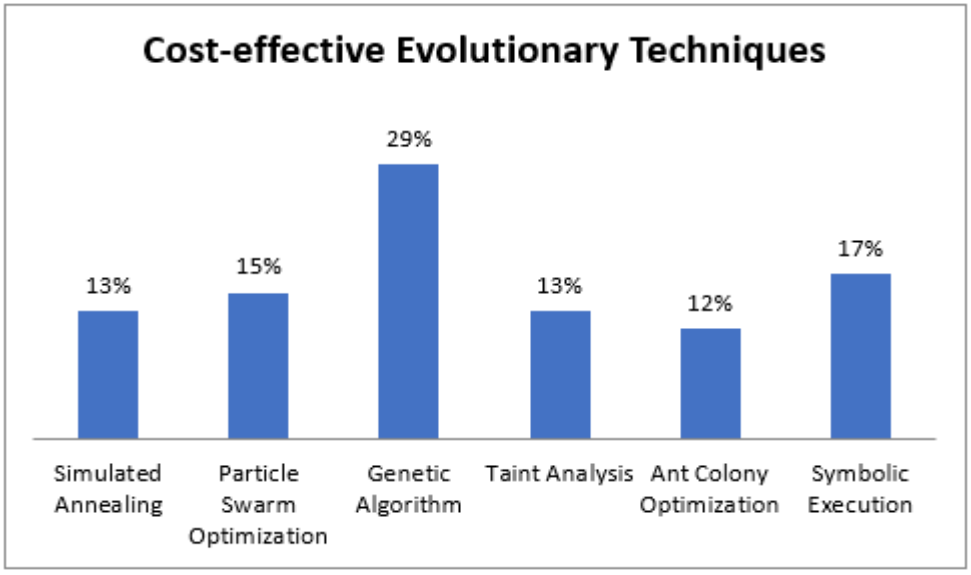

Figure 5. Cost Effective Evolutionary Testing Techniques

The findings reveal that the genetic algorithm is the most cost-effective evolutionary testing technique, as it was substantiated in 29% of the selected research articles. Following closely is symbolic execution, identified as the second most cost-effective evolutionary technique, appearing in 17% of the papers. Particle swarm optimization ranks third in cost-effectiveness, with 15% of the papers corroborating its efficacy. Taint analysis and simulated annealing constitute the fourth most frequently cited cost-effective techniques, each mentioned in 13% of the papers. Ant colony optimization, although less mentioned, is still utilized in 12% of the research articles selected for this study.

Research Question 2.2 (RQ 2.2): Which evolutionary testing technique is more scalable across various test case generations for security vulnerabilities?

This question pertains to the scalability of different evolutionary security testing techniques, focusing on their adaptability across diverse test case generations and varying security vulnerabilities. Scalability, in this context, refers to the ability of security testing techniques to adjust and accommodate different test cases, test levels, and vulnerabilities.

For this study, through systematic literature review (SLR), the scalability aspect of evolutionary testing techniques is assessed based on insights gathered from various research articles. These articles explore different techniques for security testing and conclude which ones are more scalable for different test cases, test levels, and vulnerabilities. The scalability of a particular evolutionary testing technique is evaluated based on its frequency of appearance in research articles and the authors' conclusions regarding its scalability. Based on the findings, the most scalable evolutionary security testing techniques are as follows:

Table 6. Identified Scalable Evolutionary Testing Techniques

Scalable Evolutionary Testing Technique	Reference Papers	Total Papers	%
Particle Swarm Optimization	[59], [62], [65], [69], [70], [84], [85], [88], [90], [93]	10	19%
Simulated Annealing	[52], [60], [62], [68], [70], [85], [88], [92]	8	15%
Genetic Algorithm	[52], [60], [61], [62],[65], [66], [68], [71], [78], [82], [84], [90], [91], [92], [93], [56], [77], [78], [67]	19	37%
Taint Analysis	[69], [79], [88]	3	6%
Ant Colony Optimization	[62], [64], [85]	3	6%
Symbolic Execution	[70], [71], [76], [80], [85], [87], [89]	7	13%

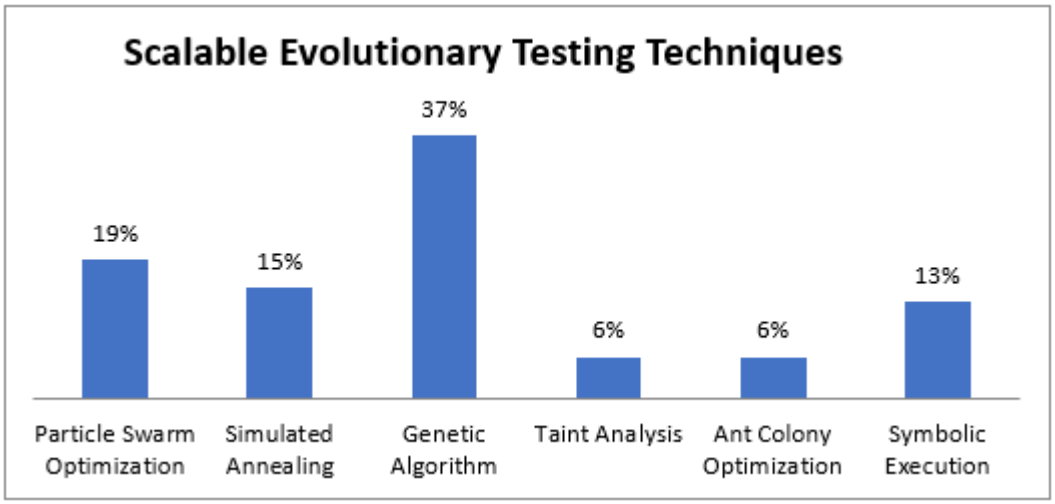

Figure 6. Scalable Evolutionary Testing Techniques

The findings suggest that the genetic algorithm is the most scalable evolutionary testing technique, as it was supported by 37% of the selected research articles. Following closely is particle swarm optimization, identified as the second most scalable evolutionary technique, appearing in 19% of the papers. Simulated

annealing emerges as the third most prevalent scalable evolutionary technique, with 15% of the papers confirming its effectiveness. Symbolic execution ranks fourth in terms of scalability, mentioned in 13% of the papers. Ant colony optimization and taint analysis were mentioned the least, yet utilized by 6% of the research articles selected for this study.

Research Question 2.3 (RQ 2.3): Which evolutionary testing technique is more effective and efficient for different types of software security testing faults?

This question determines which evolutionary testing techniques are more efficient and effective across various test levels, cases, and vulnerabilities. The answer to this question is derived from the findings of previous inquiries. To measure effectiveness and efficiency, a mathematical equation is formulated as follows:

$$EE = (Percentage\ of\ Cost - effectiveness + Percentage\ of\ Scalability)/\ 2$$

This equation evaluates effectiveness and efficiency based on the scalability and cost-effectiveness of different testing techniques. These variables are chosen because they reflect effectiveness and efficiency. A testing technique that is scalable and cost-effective is considered adequate and efficient. The percentages representing cost-effectiveness and scalability of different testing techniques are added and divided by 2 to calculate the overall percentage of effectiveness and efficiency. The two variables are divided by '2' averages, cost-effectiveness and scalability.

Table 7. EE values of evolutionary testing techniques

Efficient and Effective Evolutionary Testing Technique	Cost-effectiveness	Scalability	Result
Particle Swarm Optimization	15%	19%	17.00%
Genetic Algorithms	29%	37%	33.00%
Simulated Annealing	13%	15%	14.00%
Taint Analysis	13%	6%	9.50%
Ant Colony Optimization	12%	6%	9.00%
Symbolic Execution	17%	13%	15.00%

Therefore, when developers choose different testing techniques, they can base their selection solely on cost-effectiveness or scalability. Alternatively, they may consider the combined 'EE' factor to assess both factors or the overall effectiveness and efficiency of the testing technique. The graph below is a visual representation of the results:

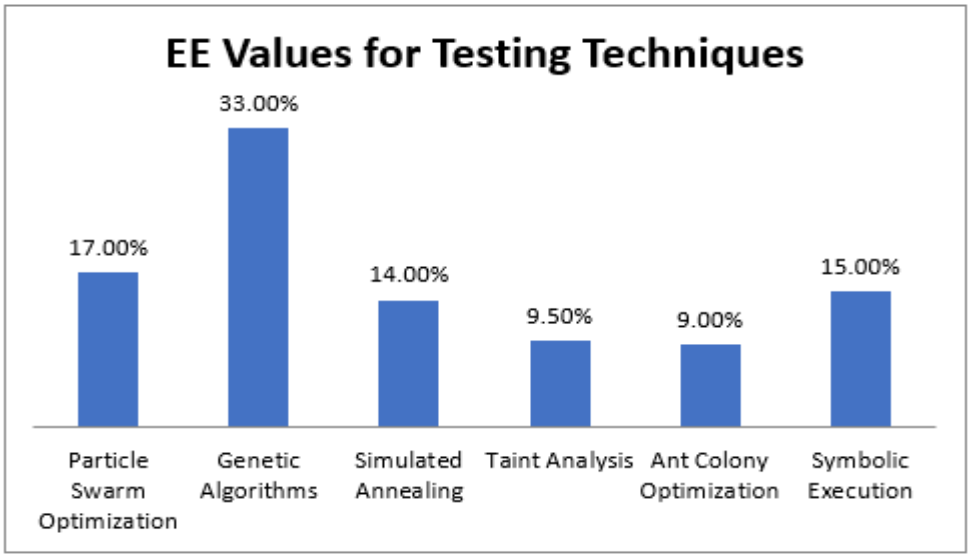

Figure 7. EE Values for Software Security Testing Techniques

The results reveal that the genetic algorithm has the highest EE value of 33%, making it the most efficient and effective evolutionary technique. The particle swarm optimization is followed closely, ranking as the second most efficient and effective testing technique with an EE value of 17%. Symbolic execution follows next with an EE value of 15%, while simulated annealing holds a value of 14%. Taint analysis and ant colony optimization rank lower in effectiveness and efficiency, with EE values of 9.50% and 9%, respectively.

These EE values generally assess the effectiveness and efficiency of various evolutionary software security testing techniques. However, it is essential to note that this indicator may not be universally applicable across all testing levels, cases, vulnerabilities, and purposes. Developers can use the EE indicator to identify suitable security testing techniques from the list provided. Nonetheless, before relying solely on the EE values, they must carefully analyze their specific test requirements, including levels, cases, vulnerabilities, and other factors. By ensuring alignment with their testing needs, developers can leverage the findings of this research and EE values to make informed decisions when selecting a testing technique.

6. Conclusion and Recommendations

Evolutionary software security testing techniques, also known as MHS algorithms, play a crucial role in enhancing the security of software systems within companies, thereby contributing significantly to the global economy. Through a systematic literature review, this study has assessed the methodologies and practices developers adopt to implement these security techniques effectively. Out of numerous articles reviewed, 52 primary studies published between 2010 and 2020 were selected for evaluation. These studies were analyzed based on various factors such as publication frequency, research methodology, adopted practices, and techniques employed for software security. The results underscore the value of systematic literature reviews in empirical research, highlighting the absence of definitive methodologies universally applicable to evaluate the effectiveness and efficiency of software security testing techniques. Instead, the suitability of methodologies or techniques varies depending on specific cases or scenarios.

The findings reveal prevalent software security vulnerabilities observed in software systems, including cross-site scripting XSS, buffer overflow/stack overflow, SQL/XML injection, privileges, permission and access control, spoofing, broken algorithms, and DoS. Similarly, a range of MHS algorithms used by developers for software security testing was identified, such as hill climbing, greedy algorithm, symbolic execution, differential evolution technique, bat algorithm, and firefly algorithm, among others. Notably, genetic algorithm, particle swarm optimization, simulated annealing, symbolic execution, taint analysis, and ant colony optimization emerged as the most utilized MHS algorithms, recognized for their scalability, cost-effectiveness, and efficiency.

Companies must continually enhance their software security techniques to safeguard their systems and maintain a competitive edge in the ever-evolving business landscape. Managers are urged to prioritize flexibility in adopting security measures tailored to their needs and operating environments. The investment in implementing these techniques is a crucial safeguard, providing robust security measures and benchmarks to fortify systems against potential threats.

6.1 Recommendation: Model for Utilizing Evolutionary Software Security Testing Techniques

Based on the findings of this systematic literature review, a model is proposed to aid developers in selecting and deploying MHS algorithms or evolutionary software security testing techniques. This model is designed for general application, guiding the selection and utilization of MHS algorithms based on identified security requirements. Factors such as scalability, cost-effectiveness, and overall efficiency and effectiveness of MHS algorithms are considered in the model.

For instance, once developers have finalized their software security testing requirements based on identified vulnerabilities, they can use the model to select the most suitable MHS algorithm or evolutionary technique. However, it is essential to note that the model's applicability is contingent upon identifying security vulnerabilities and confirming the suitability of enlisted MHS algorithms for developers' specific needs. While the model is a practical guideline, its effectiveness hinges on adherence to predefined requirements and constraints. The step-by-step model facilitates developers in effectively selecting and utilizing MHS algorithms for security testing, promoting enhanced software security across various systems.

6.2 Future Work

Based on the research findings, future studies could focus on developing a model tailored to utilize

specific MHS algorithms for addressing distinct software security vulnerabilities. This endeavor would further enrich and expand upon the insights gained in this study. Additionally, there is potential for future research to formulate methodologies outlining the practical application of MHS algorithms or evolutionary software security testing techniques. While this study has outlined prevalent software security vulnerabilities in general, a logical progression for future investigations would involve identifying which vulnerabilities manifest in specific software systems. Nevertheless, this research lays a foundational framework for future inquiries and offers practical insights beneficial to developers and researchers.

References

1. Juwita, O. and Arifin, F. N. "Design of information system development strategy based on the conditions of the organization," 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT), Kuta Bali, 2017, pp. 1-5.
2. Yasin, A.; Liu, L.; Li, T.; Fatima, R. and Jianmin, W. "Improving software security awareness using a serious game", IET Software,2019, vol. 13, no. 2, pp. 159-169.
3. Rahman, A. A. U. and Williams, L. "Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices," IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), Austin, TX, 2016, pp. 70-76.
4. Amoroso, E. "Recent Progress in Software Security," in IEEE Software, 2018, vol. 35, no. 2, pp. 11-13.
5. Khalil, M. I. K. Taj, T. "Factors Affecting the Efficacy of Software Development: A Study of Software Houses in Peshawar, Pakistan," International Review of Basic and Applied Sciences, 2021, vol. 9, no. 3, pp. 385-393.
6. Mirakhorli, M.; Galster, M.; and Williams, L. "Understanding Software Security from Design to Deployment", ACM SIGSOFT Software Engineering Notes, 2020, vol. 45, no. 2, pp. 25-26.
7. Ahmed, Z.; and Francis, S. "Integrating Security with DevSecOps: Techniques and Challenges", 2019 International Conference on Digitization (ICD), 2019, pp. 16-23.
8. Chaabouni, N.; Mosbah, M.; Zemmari, A. Sauvignac, C.; and Faruki, P. "Network Intrusion Detection for IoT Security Based on Learning Techniques", IEEE Communications Surveys & Tutorials, 2019, vol. 21, no. 3, pp. 2671-2701.
9. Sofokleous, A.; and Andreou, A. "Automatic, evolutionary test data generation for dynamic software testing", Journal of Systems and Software,2008, vol. 81, no. 11, pp. 1883-1898.
10. Alberts, J. "Integrating Threat Modeling with the SERA Method", Resources.sei.cmu.edu, 2019. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=546023. [Accessed: 2- Jun- 2019].
11. Khalil, M. I. K. Ullah, A.; Taj, A.; Khan, I.A.; Ullah, F.; Taj, F.; and Shah, S. "Analysis of Critical Risk Factors Affecting Software Quality: A Study of KPK, Pakistan Software Industry," International Review of Basic and Applied Sciences, 2022, vol. 10, no. 2, pp. 338-348.
12. Felderer, M.; Zech, P.; Breu, R.; Büchler, M.; and Pretschner, A. "Model-based security testing: a taxonomy and systematic classification", Software Testing, Verification and Reliability,2015, vol. 26, no. 2, pp. 119-148.
13. Han, T. et al., "A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers", Concurrency and Computation: Practice and Experience, 2019, vol. 32, no. 16, pp. 23-45.
14. Ferrer, F. Chicano and Alba, E. "Estimating software testing complexity", Information and Software Technology, 2013, vol. 55, no. 12, pp. 2125-2139.
15. Gordover, M. "Throwback Hack: The Citigroup Hack of 2011", ObserveIT, 2015. [Online]. Available: https://www.observeit.com/blog/throwback-hack-citigroup-hack-2011/. [Accessed: 02- Sep- 2020].
16. Vries, S. de "Software Testing for security", Network Security, 2007, vol. 27, no. 3, pp. 11-15
17. Saqib. A; Ullah, M.; Hyder, S.; Khatoon, R. and Khalil, M.I.K. "Creative Decision Making in Leaders: A Case of Beer Game Simulation," Abasyn Journal of Social Sciences, 2020, vol. 12, no. 2, pp. 379-387.
18. Mottahir, M.; Irshad, A.; and Zafar, A. "A Secure Framework for Software Product Line Development", International Journal of Computer Applications, 2017, vol. 159, no. 4, pp. 33-40.
19. Herath, T.; and Rao, H. "Encouraging information security behaviors in organizations: Role of penalties, pressures and perceived effectiveness", Decision Support Systems, 2009, vol. 47, no. 2, pp. 154-165.
20. He and Liu, Y. "Research on Software Testing to Ensure Web Application Usability, Reliability and Security", Advanced Materials Research, 2014, vol. 1049-1050, pp. 1972-1976.
21. Chen, Y. LU and XIE, X. "Research on Software Fault Injection Testing", Journal of Software, 2009, vol. 20, no. 6, pp. 1425-1443.
22. Holeňa, M. "Fuzzy hypotheses testing in the framework of fuzzy logic", Fuzzy Sets and Systems, 2004, vol. 145, no. 2, pp. 229-252.
23. Khan, I.A; Ullah, F.; Abrar, M.; Shah, S.; Taj, F. and Khalil, M.I.K. "Ransomware Early Detection Model using API-Calls at Runtime by Random Decision Forests," International Review of Basic and Applied Sciences, 2022, vol. 10, no. 2, pp. 349-359.
24. Newson, "Network threats and vulnerability scanners", Network Security, 2005, vol. 2005, no. 12, pp. 13-15.
25. Almendros-Jiménez and Becerra-Terón, A. "Automatic property-based testing and path validation of XQuery programs", Software Testing, Verification and Reliability, 2017, vol. 27, no. 1-2, p. e1625.
26. Kim, Y. and Cha, S. "Threat scenario-based security risk analysis using use case modeling in information systems", Security and Communication Networks, 2011, vol. 5, no. 3, pp. 293-300.
27. Sharma, A.; Patani, R.; and Aggarwal, A. "Software Testing Using Genetic Algorithms", International Journal of Computer Science & Engineering Survey, 2016, vol. 7, no. 2, pp. 21-33.
28. Wang, T.; Wei, T.; Gu, G.; and Zou, W. "Checksum-Aware Fuzzing Combined with Dynamic Taint Analysis and Symbolic Execution", ACM Transactions on Information and System Security, 2011, vol. 14, no. 2, pp. 1-28.
29. Yang, T.; Qian, K.; Li, L.; Lo, D.; and Tao, L. "Static Mining and Dynamic Taint for Mobile Security Threats Analysis", IEEE International Conference on Smart Cloud (SmartCloud), 2016, vol. 10, no. 5, pp. 26-38.
30. Jan, S; Maqsood, I.; Ahmad, I.; Ashraf, M.; Khan, Q.F. and Khalil, M.I.K. "A Systematic Feasibility Analysis of User Interface for Illiterate Users," Proceeding of the Pakistan Academy of Sciences, 2020, vol. 56, no. 4, pp. 2518-4253.

31. Papadakis, M. and Malevris, N. "Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing", Software Quality Journal, 2011, vol. 19, no. 4, pp. 691-723.
32. Chen, T.; Zhang, X.; Guo, S.; Li, H. and Wu, Y. "State of the art: Dynamic symbolic execution for automated test generation", Future Generation Computer Systems, 2020, vol. 29, no. 7, pp. 1758-1773.
33. Dong, C.N. and XU, B. "Effectively Metamorphic Testing Based on Program Path Analysis", Chinese Journal of Computers, 2009, vol. 32, no. 5, pp. 1002-1013.
34. Zhou, Z.; Xiang, S. and Chen, T. "Metamorphic Testing for Software Quality Assessment: A Study of Search Engines", IEEE Transactions on Software Engineering, 2016, vol. 42, no. 3, pp. 264-284.
35. Bujok, P.; Tvrdík, J. and Poláková, R. "Comparison of nature-inspired population-based algorithms on continuous optimisation problems", Swarm and Evolutionary Computation, 2019, vol. 7, no. 2, pp. 21-33.
36. Avancini, A. and Ceccato, M. "Comparison and integration of genetic algorithms and dynamic symbolic execution for security testing of cross-site scripting vulnerabilities", Information and Software Technology, 2013, vol. 55, no. 12, pp. 2209-2222.
37. Prashanta S. and Upulee K, "Fault Detection Effectiveness of Source Test Case Generation Strategies for Metamorphic Testing", Metamorphic Testing (MET) 2018 IEEE/ACM 3rd International Workshop, 2018, pp. 2-9.
38. Banković, Z.; Stepanović, D.; Bojanić, S. and Nieto, T, O. "Improving network security using genetic algorithm approach", Computers & Electrical Engineering, 2007, vol. 33, no. 5-6, pp. 438-451.
39. Petersen, K.; Vakkalanka, S. and Kuzniarz, L. "Guidelines for conducting systematic mapping studies in software engineering: An update", Information and Software Technology, 2015, vol. 64, pp. 1-18.
40. Khari, M.; Vaishali and Kumar, M. "Analysis of software security testing using metaheuristic search technique," International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2016, pp. 2147-2152.
41. Srivastava, P. R.; Ramachandran, V.; Kumar, M.; Talukder, G.; Tiwari, V. and Sharma, P. "Generation of test data using meta heuristic approach," TENCON 2008 - 2008 IEEE Region 10 Conference, Hyderabad, India, 2008, pp. 1-6.
42. Cai, K. Y. and Card, D. "An analysis of research topics in software engineering - 2006," Journal of Systems and Software, 2018, vol. 81, p. 8.
43. Tegegne, E.; Seppänen, P. and Ahmad, M. "Software development methodologies and practices in start-ups", The Institution of Engineering and Technology Journal, 2019, vol. 13, no. 6, pp. 497-509.
44. Vaishali, M. and Kumar, M. "Search-Based Secure Software Testing: A Survey", Software Testing Verification and Reliability (STVR), 2019, pp. 375-380.
45. Garn, B.; Kapsalis, I.; Simos, D. and Winkler S. 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 2014, pp. 16-21.
46. Sahin, S.; Tosun, A.; "A conceptual replication on predicting the severity of software vulnerabilities. InProceedings of the Evaluation and Assessment on Software Engineering, 2019, pp. 244-250.
47. Välja, M.; Korman, M.; and Lagerström R. A study on software vulnerabilities and weaknesses of embedded systems in power networks. InProceedings of the 2nd Workshop on Cyber-Physical Security and Resilience in Smart Grids, 2017, pp. 47-52.
48. Pham, N.; Nguyen, T.; Nguyen, H.; Nguyen, T. Detection of recurring software vulnerabilities. InProceedings of the IEEE/ACM international conference on Automated software engineering, 2010, pp. 447-456.
49. Anand, P.; Ryoo, J. "Architectural Solutions to Mitigate Security Vulnerabilities in Software Systems. InProceedings of the 13th International Conference on Availability, Reliability and Security, 2018, pp. 1-5.
50. Kenner, A. "Model-Based Evaluation of Vulnerabilities in Software Systems. InProceedings of the 24th ACM International Systems and Software Product Line Conference-Volume B, 2020, pp. 112-119.
51. Pan, X. and Chen, H. "Using Organizational Evolutionary Particle Swarm Techniques to Generate Test Cases for Combinatorial Testing," 2011 Seventh International Conference on Computational Intelligence and Security, 2011, pp. 1580-1583.
52. Chen, Y.; Poskitt, C. M.; Sun, J.; Adepu, S. and Zhang, F. "Learning-Guided Network Fuzzing for Testing Cyber-Physical System Defences," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 962-973.
53. Saber, T.; Delavernhe, F.; Papadakis, M.; O'Neill, M. and Ventresque, A. "A Hybrid Algorithm for Multi-Objective Test Case Selection," 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1-8.
54. Khari, M.; Vaishali and Kumar, M. "Analysis of software security testing using metaheuristic search technique," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 2147-2152.
55. Khalil, M. I. K.; Shah, S. A. A.; Khan, I. A.; Hijji, M.; Shiraz, M.; and Shaheen, Q. "Energy cost minimization using string matching algorithm in geo-distributed data centers," Computers, Materials, and Continua, 2023, vol. 75, no. 3, pp. 6305-6322.
56. Baca, D.; Petersen, K.; Carlsson, B. and Lundberg, L. "Static Code Analysis to Detect Software Security Vulnerabilities - Does Experience Matter?", 2009 International Conference on Availability, Reliability and Security, 2009, vol. 7, no. 2, pp. 21-33.
57. Thakkar, A. and Lohiya, R. "Role of swarm and evolutionary algorithms for intrusion detection system: A survey", Swarm and Evolutionary Computation, 2020, vol. 53, p. 100631.
58. Avancini, A. and Ceccato, M. "Towards security testing with taint analysis and genetic algorithms", Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems - SESS ', 2010, , vol. 9, no. 1, pp. 15-30.
59. Shilpi and Karambir, "Improvising the effectiveness of test suites using differential evolution technique," 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2016, pp. 52-56.

60. Zou, D.; Wang, R.; Xiong, Y.; Zhang, L. Su, Z. and Mei, H. "A Genetic Algorithm for Detecting Significant Floating-Point Inaccuracies", 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, , vol. 2, no. 6, pp. 45-85.
61. Avancini, A. and Ceccato, M. "Security Testing of Web Applications: A Search-Based Approach for Cross-Site Scripting Vulnerabilities", 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation, 2011, , vol. 1, no. 2, pp. 8-15.
62. Ahmad, I.; Khalil, M. I. K.; and Shah, S. A. A. "Optimization-based workload distribution in geographically distributed data centers: A survey," International Journal of Communication Systems, 2020, vol. 33, no. 12, p. e4453.
63. Luckow, K.; Kersten, R. and Pasareanu, C. "Complexity vulnerability analysis using symbolic execution", Software Testing, Verification and Reliability, 2020, vol. 30, no. 7-8.
64. Avancini, A. and Ceccato, M. "Comparison and integration of genetic algorithms and dynamic symbolic execution for security testing of cross-site scripting vulnerabilities", Information and Software Technology, 2013, vol. 55, no. 12, pp. 2209-2222.
65. Păsăreanu, C. and Visser, W. "A survey of new trends in symbolic execution for software testing and analysis", International Journal on Software Tools for Technology Transfer, 2009, vol. 11, no. 4, pp. 339-353.
66. Peng, K. and Huang, C. "Stochastic modelling and simulation approaches to analysing enhanced fault tolerance on service-based software systems", Software Testing, Verification and Reliability, 2015, vol. 26, no. 4, pp. 276-293.
67. Yu, T.; Srisa, W.; Cohen, M. and Rothermel, G. "A hybrid approach to testing for nonfunctional faults in embedded systems using genetic algorithms", Software Testing, Verification and Reliability, 2015, vol. 28, no. 7.
68. Kronjee; J.; Hommersom, A.; Vranken, H. Discovering software vulnerabilities using data-flow analysis and machine learning. InProceedings of the 13th International Conference on Availability, Reliability and Security, 2018 Aug 27, pp. 1-10.
69. Salimi, S.; Ebrahimzadeh, M.; Kharrazi, M. Improving real-world vulnerability characterization with vulnerable slices. InProceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, 2020 Nov 8, pp. 11-20.
70. Khalil, M. I. K.; Ahmad, I.; Shah, S. A. A.; Jan, S.; and Khan, F. Q. "Energy cost minimization for sustainable cloud computing using option pricing," Sustainable Cities and Society, 2020, vol. 63, p. 102440.
71. Aquino, A.; Braione, P.; Denaro, G. and Salza, P. "Facilitating program performance profiling via evolutionary symbolic execution", Software Testing, Verification and Reliability, 2021, vol. 30, no. 2, pp. 12-23.
72. Păsăreanu, C.; Kersten, R.; Luckow, K. and Phan, Q. "Symbolic Execution and Recent Applications to Worst-Case Execution, Load Testing, and Security Analysis", Advances in Computers, 2019, pp. 289-314.
73. Ande, R.; Adebisi, B.; Hammoudeh, M. and Saleem, J. "Internet of Things: Evolution and technologies from a security perspective", Sustainable Cities and Society, 2020, vol. 54, p. 101728.
74. Muhammad, D.; Ahmad, I.; Khalil, M. I. K.; Khalil, W.; and Ahmad, O. A. "A generalized deep learning approach to seismic activity prediction," Applied Sciences, MDPI, 2023, vol. 13, p. 1698.
75. Alhazmi, O.; Malayia, Y. and Ray, I. "Measuring, analyzing and predicting security vulnerabilities in software systems", Journal of Systems and Software (JSS), 2007, vol. 26, no. 3, pp. 219-228.
76. Garousi, V. "A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation," in IEEE Transactions on Software Engineering, 2010, vol. 36, no. 6, pp. 778-797.
77. Ali, S.; Briand, L. C.; Hemmati, H. and Panesar, W R. K., "A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation," in IEEE Transactions on Software Engineering, 2010, vol. 36, no. 6, pp. 742-762.
78. Han, T. et al., "A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers", Concurrency and Computation: Practice and Experience, 2019, vol. 32, no. 16, pp. 45-56.
79. Yoo, S. and Harman, M. "Regression testing minimization, selection and prioritization: a survey", Software Testing, Verification and Reliability, 2012, vol. 22, no. 2, pp. 67-120.
80. Baca, D.; Carlsson, B.; Petersen, K. and Lundberg, L. "Improving software security with static automated code analysis in an industry setting", Software: Practice and Experience, 2012, vol. 43, no. 3, pp. 259-279.
81. Nayak, K.; Marino, D.; Efstathopoulos, P. and Dumitraş, T. "Some Vulnerabilities Are Different Than Others", Research in Attacks, Intrusions and Defenses, 2014, pp. 426-446.
82. Ahmad, I.; Ahmad, M. O.; Alqarni, M. A.; Almazroi, A. A.; and Khalil, M. I. K. "Using algorithmic trading to analyze short-term profitability of Bitcoin," PeerJ Computer Science, 2021, vol. 7, p. e337.
83. Singhal, A.; Bansal, A. and Kumar, A. "A critical review of various testing techniques in aspect-oriented software systems", ACM SIGSOFT Software Engineering Notes, 2013, vol. 38, no. 4, pp. 1-9.
84. Pham, N.H.; Nguyen, T.T.; Nguyen, H.A.; Wang, X.; Nguyen, A.T. and Nguyen, T.N. Detecting recurring and similar software vulnerabilities. In2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010 May 2, Vol. 2, pp. 227-230.
85. Ashraf, R. A. and DeMara, R. F. "Scalable FPGA Refurbishment Using Netlist-Driven Evolutionary Algorithms," in IEEE Transactions on Computers, 2013, vol. 62, no. 8, pp. 1526-1541.
86. Khalil, M. I. K.; Shah, S. A. A.; Taj, A.; Shiraz, M.; Alamri, B.; Murawat, S.; and Hafeez, G. "Renewable aware geographical load balancing using option pricing for energy cost minimization in data centers," Processes, MDPI, 2022, vol. 10, no. 10, p. 1983.
87. Jan, S.; Panichella, A.; Arcuri, A. and Briand, L. "Automatic Generation of Tests to Exploit XML Injection Vulnerabilities in Web Applications," in IEEE Transactions on Software Engineering, 2019, vol. 45, no. 4, pp. 335-362.

88. Tokar, J.L.; Jones, F.D.; Black, P.E.; Dupilka, C.E. Software vulnerabilities precluded by spark. InProceedings of the 2011 ACM annual international conference on Special interest group on the ada programming language, 2011, pp. 39-46.
89. Khalil, M. I. K.; Ahmad, A.; Almazroi, A.A. "Energy Efficient Workload Distribution in Geographically Distributed Data Centers," IEEE Access, 2019, vol. 7, no. 1, pp. 82672-82680.
90. Alboaneen, D. et al. "A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers." Future Generation Computer Systems, 2021, pp. 201-212.
91. Humayun, M.; Niazi, M.; Jhanjhi, N.; Alshayeb, M. and Mahmood, S. "Cyber Security Threats and Vulnerabilities: A Systematic Mapping Study", Arabian Journal for Science and Engineering, 2020, vol. 45, no. 4, pp. 3171-3189.
92. Sparks, S.; Embleton, S.; Cunningham, R. and Zou, C. "Automated Vulnerability Analysis: Leveraging Control Flow for Evolutionary Input Crafting," Twenty-Third Annual Computer Security Applications Conference, 2010, pp. 477-486.
93. Khalil, M. I. K..; Mubeen, A.; Taj, A.; Jan, N.; Ahmad, S. "Renewable and Temperature Aware Load Balancing for Energy Cost Minimization in Data Centers: A Study of BRT, Peshawar," Journal of Computing & Biomedical Informatics, 2023, vol. 06, no. 3, pp. 1-8.