Software Security Static Analysis False Alerts Handling Approaches

Aymen Akremi
College of Computer and Information Systems
Umm Al-Qura University (UQU)
Makkah, Saudi Arabia

***Abstract*—False Positive Alerts (FPA), generated by Static Analyzers Tools (SAT), reduce the effectiveness of the automatic code review, letting them be underused in practice. Researchers conduct a lot of tests to improve SAT accuracy while keeping FPA at a lower rate. They use different simulated and production datasets to validate their proposed methods. This paper surveys recent approaches dealing with FPA filtering; it compares them and discusses their usefulness. It also studies the used datasets to validate the identified methods and show their effectiveness to cover most program defects. This study focuses mainly on the security bugs covered by the datasets and handled by the existing methods.**

Keywords*—*Software security; static analysis; false alert reduction; source code dataset; security bugs

I. Introduction

Software coding and implementation have grown fastly during the last years. This is due to the rapid migration towards bits and the extensive use of digital technologies. The more software applications become relevant, the more security assurance of programs gets essential. However, software security defects increased due to implementation failures regarding security best coding practices. Escaping software faults into later stages of software development will increase the maintenance cost[1],[2]. Also, after application deployment, cyberhackers will try to detect these coding vulnerabilities and exploit them to achieve their goals. Thus, coding review and auditing is a primordial task before software use.

Static Analysis Tools (SAT) play an essential role in automatically detecting these vulnerabilities and alerting the programmer, which reduces the auditing time, effort, and cost. SAT automatically examines the code for any programming defects without executing the code and generates alerts about possible errors. Alerts provide the auditor with useful information such as the location of the purported defect in the source code, the nature of the fault, and additional contextual information. However, the SAT still suffers from several issues, letting them underused in practice. Among them, this study focuses on the large number of warnings generated by SAT; most of them are false positives, which is a time-consuming and painstaking task to review them all.

One approach to deal with a large number of FPAs is by unsoundly processing source code. Almost all existing SATs are uniformly unsound [3]. Loops and unknown external libraries call, for instance, are a significant source of imprecision. Unsound SAT considers only a fixed number of loops while ignoring the rest and assumes any unknown external library call as predefined behaviors such as skip[3]. This unsoundness regarding loops and unknown external libraries causes the analysis to miss a significant amount of real bugs and reduce false-positive alerts.

In this study, any paper that sacrifices SAT soundness to reduce false-positive alerts is ignored. Ideally, an SAT must be precise and scalable while avoiding false positives.

Existing efforts dealing with the false-positive alert reduction face several challenges, mainly are:

- Handling of a large code base will decrease SAT precision; most of them perform better in a small set of problems. Besides, processing a significant codebase causes the SAT over-approximation of the input program behavior, which may consider correct program properties as errors.
- Increasing SAT precision raises much more false-positive alerts. The challenge is how to keep a high detectability rate without throwing FPAs.
- The inability of the SAT to get knowledge about the software architecture, its dependencies, and the manner of how data flows through the system, which may result in throwing FP alerts considered as potential errors [4].

So researchers are trying to solve one challenge or some of them to reduce false-positive alerts.

To our best knowledge, these different approaches have not been studied rigorously and comprehensively. Thus, the objective of this paper is the investigation of current methods dealing with false alert elimination. It mainly presents the most significant efforts in this field and their scalability in the last ten years. It defines new criteria to compare different approaches. Also, this study focuses on showing the most effective dataset used in the literature and provides statistics about them. Finally, the paper discusses the advantages and shortcomings of FPA handling approaches and presents recommendations to improve the SAT.

This paper is divided into eight parts; after introducing the research subject in Section 1, it presents the related works in Section 2. The paper shows the research methodology for selecting the relevant articles in Section 3 and existing approaches identified categories in Section 4. The paper compares, in Section 5, the different methods used to reduce false alerts. Section 6 provides an overview of the used datasets, then discusses the shortcomings and proposes recommendations to

deal with these limitations in Section 7. Finally, the paper is concluded in Section 8.

II. Related Works

In paper [5], authors studied the existing efforts aiming at combining static analysis and dynamic quality assurance techniques to improve SAT bugs detection with reduced false alerts. They finally selected 51 articles for their mapping study. Thus, they include only papers that consist of the integration of combined technologies so that the output of one method is the input of the second. However, this paper shows the different approaches categories and any possible combination used to improve SAT precision or reduce FP alerts.

Heckman et al in [6] investigates 18 research effort to identify actionable alert identification techniques. They categorize the approaches as classification or ranking methods. The authors also conducted a comparative study to identify the approaches having the best accuracy. In this effort, articles that improve SAT precision to reduce FP alerts, not only improving the bugs detection rate, are also studied.

Similar as [5], authors in the paper [7] identified 51 papers for their mapping study. They focus on the study of the existing static analysis tools and techniques to reduce false alerts. However, this article covers only methods handling false alerts.

The paper [8] surveys 79 articles that handle the enormous amount of FP alerts after their generation. The authors focus on the methods dealing with the reduction of SAT alert reports. While, this study considers all kinds of unique approaches that help minimize FP alerts, whether the method is for the refinement of the software source code, the improvement of SAT precision, or the post-handling of SAT alerts report.

It is worthy to note that all the reviewed papers by the above surveys were published four years ago since the last study [8] at our best knowledge published in 2016. Thus, this effort focuses on the recent papers fitting the selection requirement as maximum to provide researchers with a recent and accurate literature review.

This study outperforms the above surveys by:

- the selection and presentation of relevant datasets to test and validate the SAT tools. It collects the different open source datasets along with information about their size and features (see Section VI).
- the presentation of the features used by the identified methods for their model training and alerts prediction or classification(see Section V).
- providing the reader with the different types of security bugs handled by the identified approaches alongside with the paper reference (see Section VI-B).
- the comparison of the different false alert handling techniques according to their scalability in order to study their ease of integration and application (see Section V).
- depicting ongoing projects and competition aiming at boosting the researches to improve SATs and at providing accurately labeled datasets (more details in VI).

TABLE I. Search Keywords Categories

Category Number	Keywords
1	defects, bugs, faults
2	false alerts, false warnings, false alarms
3	static analysis, source code analysis, automatic static bugs detection
4	filtering, elemination, reduction, handling

Several other existing studies, such as [9], [10], [11], [12], [13], evaluate the SAT in terms of precision and alert handling and conduct a comparison study between them. This paper has a different objective by only presenting the approaches that improve SAT alerts handling, not testing their precision.

III. Research Methodology

This survey starts by identifying relevant papers that deal with false alert reduction. Fig. 1 depicts the main steps to select pertinent articles and extract information from them.

A. Research Questions

The process of relevant paper selection goes through the precise definition of the research topic, enabling identifying the keywords used for the scientific database search. This study aims at answering the following questions:

- ***RQ.1*: What are the different techniques used to reduce FPAs?
- ***RQ.2*: How extend human effort is required to execute the proposed approach?
- ***RQ.3*: Are the proposed approaches scalable?
- ***RQ.4*: Are security bugs considered during the FPAs reduction?
- ***RQ.5*: What are the datasets used to validate the different methods?

B. Used keywords and Search Engine Configuration

The relevant keywords are determined based on the research questions identified in Section III-A. Keywords could be classified into four categories representing the most used terms and their synonyms. Then for each search round, a combination of keywords taken from each set is used. The used keywords are listed in Table I.

The first category encompasses the most used names of program errors. The second category contains the different terms of alerts; more specifically, it focuses on false-positive alerts. The third category includes possible static analysis names that different researchers may use, and finally, the last category contains the used keywords to describe alert reductions.

So, this study makes 108 = 3 × 3 × 3 x 4 separate search strings rounds at Google scholar, which ranks research papers based on their relevance. It refines the search by showing only articles published after 2010 to ensure that the selected documents consider recent programming technologies and new trends of SATs. The first 50 papers that match all the searched keywords combinations are chosen. So, this paper identified 540 articles before proceeding with the selection process.

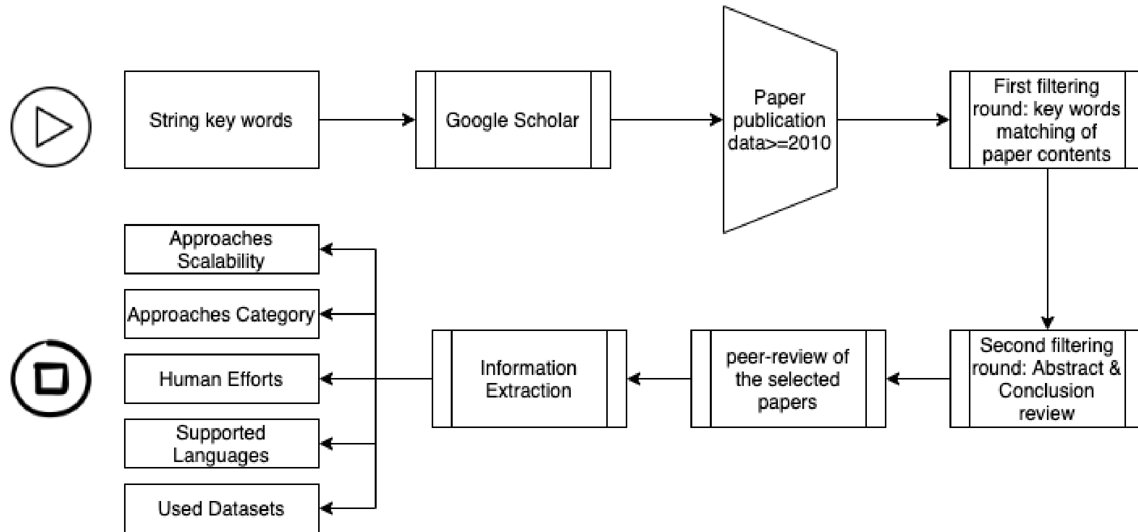

Fig. 1. Research Methodology Diagram.

C. Relevant Papers' Selection Process

This section presents the paper selection process that consists mainly of the quick and peer review of the candidate articles from the previous steps. Papers are filtered quickly at the second filtering round based only on the title, abstract, evaluation, and conclusion. Only papers satisfying the following criteria are included in the final peer review:

- papers that explicitly aim to reduce false alerts. Thus, any effort based on improving the precision of the static analyzer or modifying the software source code, or post handling of SAT alert reports is included.
- papers that have an evaluation and test of their approach.

Also, this study excludes papers that:

- sacrifices the soundness of the SAT to reduce false-positive alerts.
- aims only to detect true positive alerts without reducing FPAs.
- only surveys existing efforts without providing any new technique or approach to reduce FPAs.
- mostly uses similar techniques and datasets to another already selected paper. The aim is to keep the uniqueness and originality of each chosen article.

After this process, 30 relevant articles that summarize almost all approaches and efforts dealing with SAT false alert handling are finally selected. The distribution number of chosen papers according to the Scientific publisher databases are shown in Table II

TABLE II. DISTRIBUTION OF THE SELECTED PAPERS ACCORDING TO THE PUBLISHERS

Publisher	# of papers	Journal/Conference name
hal.archives-ouvertes.fr	1	10th European Congress on Embedded Real Time Software and Systems 2020
ScienceDirect	2	Journal of Systems and Software 137 (2018): 766-783 Information and Software Technology 52.2 (2010): 210-219
Springer	6	Asian Symposium on Programming Languages and Systems. Springer, Cham, 2014 IFIP Int. Conference on Open Source Systems. Cham, 2018 Int. Static Analysis Symposium.Berlin, Heidelberg, 2016 Int. Conference on Software Analysis, Testing, and Evolution. Cham, 2018 Int. Symposium on Formal Methods.Cham, 2015 OTM 2017 Conferences, Part II, LNCS 10574, pp. 99–106, 2017
ACM	8	ACM Transactions on Programming Languages and Systems, Vol. 39, No. 4, 2017 15th Int. Symposium on Open Collaboration. 2019. 33rd Annual Computer Security Applications Conference. 2017 27th ACM SIGSOFT international symposium on software testing and analysis. 2018 27th Annual ACM Symposium on Applied Computing. 2012 ACM on Programming Languages 1.OOPSLA (2017): 1-30-journal 40th Int. Conference on Software Engineering: Companion Proceedings MAPL'17, June 18, 2017, Barcelona, Spain - conference
IEEEXPLOR	13	26th Int. Symposium on Software Reliability Engineering (ISSRE) 12th IEEE Conference on Software Testing, Validation and Verification (ICST).2019 27th Int.Symposium on Software Reliability Engineering.2016 6th Int.Workshop on Software Engineering Research and Industrial Practice. 2019 41st Int. Conference on Software Engineering: Software Engineering in Practice.2019 10th Int. Conference on Fuzzy Systems and Knowledge Discovery (FSKD). 2013 15th Int.Conference on Computer Systems and Applications (AICCSA). 2018 Formal Methods in Computer Aided Design., 2010 39th Int.Conference on Software Engineering (ICSE). 2017 Int. Conference on Big Data (Big Data),2018 1st Int. Workshop on Software Qualities and their Dependencies (SQUADE). 2018. 38th Int. Conference on Software Engineering Companion (ICSE-C).2016. 2014 21st Asia-Pacific Software Engineering Conference

D. Information Extraction Process

In this step, this study proceeds for peer review of the identified papers to extract the relevant and targeted information, which are:

- the used approaches or techniques.
- the application level of the approach. It means if the proposed method deals with improving the precision of SAT or modifying the software source code before analyzing it, or post handling of SAT reports.
- the coverity of the approaches to detect most programming bugs since several articles only reduce false alerts generated by specific bugs.
- the human intervention effort during the false alert filtering.
- the FPAs reduction percentage, whether explicitly mentioned or could be deduced from the other metrics presented in the paper. In some articles, it is not possible to extract the FPA reduction rate due to the

lack of specific measures.

- the programming language of the examined application.
- the SAT used for code examination.
- the dataset used to evaluate the proposed approach.

All gathered information is carefully saved in an Excel sheet database created to facilitate their mining. The extracted data contains the required information to answer this study's research questions.

IV. False Alerts Handling Approaches: A Classification

To answer RQ1, the paper starts by identifying the used approach of each article and categorizes them based on the similarities of the used techniques. This study distinguishes mainly seven categories, as shown in Fig. 2, which are: Machine Learning (ML) based approaches, Root Causes (RC) based approaches, Model Checking (MC) based approaches, Data Mining (DM) based approaches, and Semantics (SM) based approaches, Rule(RU) based approaches, and Slicing (SG)Based approaches.

A. Machine Learning-based Approaches

Machine Learning is the science of teaching a computer how to learn from data and create a model used after that to predict/classify new data[14]. It works mainly with algorithms, not raw data. ML is widely used in the field of static analysis to improve the SAT precision or post-handle the SAT-generated alarms and predict their truthness (resp. falseness).

Authors in [15], [16] have similar works that consist of establishing a new classifier based on additional learning features, which is the program structure patterns that correlate similar false alarms. They use mainly Naïve Bayes, LSTM (long-short term memories), and SVM to predict new alerts. In [17]and [18], the authors propose a clustering-based approach to classify and correlate similar alerts generated from the SAT. They formalize new methods to find dependencies between alarms caused by the buffer overflow error. Then, they cluster dependent warnings in the same cluster. After that, they tag the groups based on the dominant sound alerts. In [19], authors train a decision tree ML technique using ensemble learning (i.e.training several weak classifiers to form a new combined stronger model; authors use AdaBoost for ensemble learning) to classify alerts. They labeled the training dataset generated from multiple SATs to train the created model. Their approach is based only on the SAT reports, which provide their solution better scalability(no code pre-processing is required to try their approach) Authors in [20] proposed an approach that merges several SAT alerts to extract features used in the prediction model. They use four machine learning techniques to identify the best reducing false alarms. The paper [3] tries to deal with unsoundness static analysis and the tradeoff between False Negative rate (FNR) and False Positive Rate (FPR). Since reducing FPR increases the FNR, which is more critical and vise versa. They proposed to selectively learn their SVM model by only harmless codeset structures used to predict only FPAs. In [21] and [22] authors uses ML techniques to reduce false alarms. They use typestate variables and software engineering metrics to learn their model and predict false alerts.

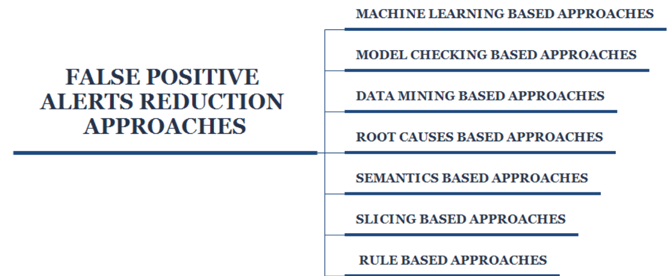

Fig. 2. False Positives Alerts Reduction Approaches.

Authors in [23] use lexical tokenization labeled by the human to learn their CNN classifier to reduce false alerts. They propose a continuous mechanism for code integration after review.

B. Root Causes based Approaches

Root causes analysis is the process of identifying and investigating the causes of events occurrences. Therefore, investigators could specify effective corrective measures [24]. This technique is used to identify SAT false alerts root causes to eliminate or filter them. In [25], authors conduct a manual inspection of 30 javascript web application alerts generated by the static analyzer, and they conclude seven root causes of alarms. Then they use a different technique for each identified root cause to eliminate any generated alert. Authors[26] aims to reduce false alerts by reporting to the SAT user the alarm root causes to be inspected instead of the alarm itself. Also, they ask the user to answer questions related to the root causes to fix the error until no more alarm is triggered. Their approach requires extensive interaction with humans to validate root causes and define the corrective measures. The paper [27] aims to overcome the issues of the alert propagation technique. It consists of inserting new alerts before or after their causes location and removing original alarms generated by the SAT. However, the number of warnings may increase in several cases. Their paper overcomes this issue by repositioning alerts to their causes instead of creating new alerts and removing the original alert after that.

C. Model Checking based Approaches

Model Checking is a formal verification technique that investigates all possible states of a given system based on a model that defines the system behavior properties. The MC verification technique is as proper as the model representing the system [28]. SATs widely use MC techniques to reduce false-positive alerts by verifying their correctness according to the predefined model. The paper [29] aims at implementing a software analyzer that could process large-scale lines of codes with high precision at the expense of completeness and possible missing of potential defects. Their main idea is the use of specialized abstraction based on both data and predicate abstraction bounded on several model checkers. Similarly, Microsoft uses MC based static analyzer to review its software codes. Their product SLAM2 uses a model checking approach

over abstract C program statements to identify program defects and eliminate false warnings[30]. In [31], authors made a benchmark using the LABMC model checking for false alert reduction. They add loop abstraction before the use of the LABMC model checker. Authors in [32] aim to detect FPAs via the use of deductive checking to verify the conforms of source code position reported by the alert with a standard coding protocol such as Sei Cert C and ANSI/ISO. Authors in [33] aim to improve the scalability of model checking to handle the massive amount of generated SAT false positive alerts. They introduce a new variable named complete-range non-deterministic values (cnv) to reduce and avoid redundant verification calls of the model checker, mostly responsible for generating false-positive alerts. Another use of system verification techniques is the employment of Satisfiability modulo theories (SMT) solvers to identify the true/false alerts. In paper [34], authors use first abstract based analysis to fastly review codes, then link alarms to the related code snippet. After transforming alerts to SMT acceptable formulae, they use it to check the properness of such warnings.

D. Data Mining based Approaches

Data Mining (DM) techniques are used to identify hidden, potential, and valuable patterns from extensive data [35]. It is designed to extract the rules from a vast amount of data to be used by the human or other automated techniques [36]. Frequently, DM is used in combination with ML techniques that use DM-generated patterns as features to learn ML model [37]. SAT uses DM techniques to identify false-positive alert patterns for further filtering. Authors in [38] use a frequency-based algorithm to discover similar warnings patterns of SAT alerts. They transform generated warnings to composed traces and then compute their similarity using a DM-based technique that calculates similar patterns' frequencies. Then they use the patterns to filter false alerts. In [39], the authors use the Stochastic gradient descent (SGD) DM technique to reduce the complexity of finding patterns from important alerts set of several SAT's reports. Then, the authors use the Adaboost ML-based technique to create a stronger classifier trained from the SGD output.

E. Rule based Approaches

Rule-based approaches are used to manipulate knowledge to interpret information in a useful way. Rules are provided by a human or automatically generated using machine learning algorithms [38]. The latter is called Rule-based machine learning, considerably used in SAT precision enhancement and FPA reduction. Authors in [40] design and implement a bug detection software based on a set of rules extracted from manual inspection of software patches. They refine rules using a feedback-based approach by iteratively improving them each time their SAT reports a false alert. In [41], authors propose a new extension to the industrial static analyzers to fix the multiple locations of frequent warnings using experts' knowledge in the form of rules. Their expansion reduces only one false alert type by detecting the alert's name and applying a rule-based knowledge algorithm to check its truth. Authors in [4] propose a new algorithm to distinguish true positive from false-positive alerts. They try to identify the connection between the CWE and false positives to extract new rule-based patterns.

F. Semantics based Approaches

Semantic approaches refer to the meaning of language constructs. It *"provides the rules for interpreting the syntax which does not provide the meaning directly but constrains the possible interpretations of what is declared,"* according to Euzenat [42]. The semantic approach uses mathematical logic to build rules describing constructs and relations identified in the program code. In [43] use logic programming language named DataLog to build their declarative static analyzer called URSA with the help of interactive user questions to identify alarm root causes. This tool augments the semantics of DataLog to control its over-approximation. Authors in [44] define new abstract domains that specify software violations. They apply the finite state machine technique to determine these domains and use them with a semantic-based static analyzer. In paper [45], authors propose an algorithm to generate a program graph that is used along with a static analyzer report to prioritize true bugs and reduce false alerts. Their main contribution is extracting semantic information to calculate the severity level of warnings and then using the graph algorithm to prioritize SAT alerts.

G. Slicing based Approaches

The program slicing approach is mainly used to avoid the complexity analysis of codes by reducing the original program to its minimal form called slice while keeping the same program behavior [46]. It consists of the computation of a program statement set, called program slices, that may affect the values at some point of interest. The slicing approach is used widely in program debugging to locate errors more easily [47]. There exist two types of Slicing techniques: static program slicing and dynamic slicing. The first, according to the original definition of Weiser, consist of all statements in a program that may affect the value of a specific variable in a certain statement [46]. In contrast, dynamic program slicing "contains all statements that actually affect the value of a variable at a program point for a particular execution of the program rather than all statements that may have affected the value of a variable at a program point for any arbitrary execution of the program" [48].

The main idea of the slicing approach proposed by [49] is the decomposition of the program into several executable slices and run dynamic analysis over each of them, which will reduce the processing time and complexity and consequently reduce the false alarms. Authors in [50] aim to focus directly on the sliced code generated by the alarm and verify its correctness. After applying static analysis over JAVA EE code, they slice the code based on the linked alert, transform it into executable slices and verify the code again while filtering any false alarm.

V. Comparison and Analysis of Recent Existing Efforts

This section provides the different extracted data from the 30 selected papers after several rounds of peer-reviewing depicted in Table III.

This effort starts by depicting the papers processed bugs called ***bugs coverity*** aiming at knowing whether the proposed approach deals with all security bugs or just focuses on some types. According to the Table III, 53.3% of the approaches

filter all kinds of defects in general. However, a considerable effort, about 46%, focuses only on specific types of defects, and therefore they could not be used without combining with other methods. Also, only 43% of papers explicitly aim to reduce false-positive alerts while maintaining high accuracy in detecting true security bugs.

Then, the paper show the ***categorization*** of the different approaches as detailed in S IV. The extensive use of *ML based approaches* to reduce false alerts is very observable, which is very expected since ML techniques outperform other methods when treating big data. However, the main issue of ML-based approaches is the need for a large amount of labeled data to obtain satisfactory accuracy. ML-based techniques are combined with model checking methods to verify source code properties better, extract features, and predict or classify the alerts. All identified papers that use ML techniques are applied to the source code or SAT alert reports.

Data mining-based approaches are used in four papers to reduce false-positive alerts. Also, none of the identified articles using DM methods are applied to the SAT source code level. It is explained by the SAT use of verification techniques based on knowledge rules to check software source code rather than ML or DM based models.

Model Checking based approaches used logical rules to verify source code properties or alerts truthiness. MC methods are applied and used for all integration levels.

Root causes based approaches as well are used to identify the location of alert causes from the examined software source code. Thus, all papers using root causes-based approaches apply their methods to both software source code and SAT alert reports.

Semantic based approaches is generally used to extract source code properties used further as patterns and features by SAT.

Slicing based approaches most times used to reduce source code complexity by decomposing it into small slices having the same behavior then run SAT over reduced programs which improve its soundness without throwing a large number of false alerts. *Rule based approaches* are only used with software source code for rule patterns extraction used after that by ML or DM based techniques to predict or classify alerts.

The ***supported languages*** feature aims to understand the research direction focus on the handled languages. Since C language is unsafe, most SAT analyzers are dedicated to analyzing C codes. Consequently, most approaches dealing with FP alert handling are generated from the static analysis of C implemented applications.

The ***Scalability*** feature seeks to depict the extend of a proposed approach to easily being used by most users. In Fig. 3, this article distinguishes three application levels of false alert handling approaches, which are *Software source code level, Static analyzer source code, and Static analyzer alerts report*. Also, Fig. 3 summarizes each application level's most used approach categories.

This paper consider approaches dealing false alert handling only from SAT reports as the most scalable. It is explained

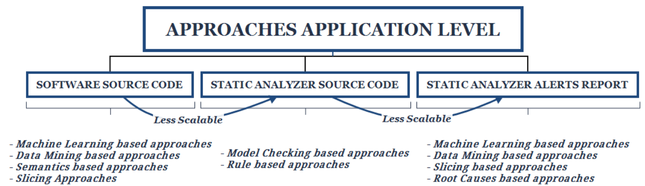

Fig. 3. Approaches Application Levels.

by the direct processing of SAT reports without any pre-processing, which will avoid any inconvenience when trying to adopt the approach. Meanwhile, approaches already integrated with SAT tools are also easy to use since the difficulty is only in the integration step already made by the approach's authors.

The ***human effort*** feature shows the approach reliance extend of human intervention. Of course, each time the proposed method does not require human interaction, it is considered more effective, scalable, and time/cost-saving. Almost all ML and DM-based approaches require moderate to extensive human intervention. This is due to the labeling effort required to train the created models. Few ML-based approaches require reduced human efforts explained by using a clustering approach to label the dataset then use it to lean an ML model. It is observable from the Table III that almost proposed approaches, that do not require human intervention, are applied to the SAT alerts report.

The ***False Alerts Reduction Rate*** feature extracts the reduction rate of false alerts, as mentioned by the paper authors. Some papers explicitly present the reduction rate while, in other articles, the FPA reduction rate is deduced. Almost approaches do not exceed 90% of reduction rate except one paper [41] that reaches a 100% reduction rate but for only one type of alert.

VI. An Overview of the Used Datasets

Finding or creating an effective dataset that reflects the real issues and complexity of software source code analysis to validate SATs is of paramount importance. To facilitate the identification of valuable datasets, this study extracts the relevant datasets used by the selected papers and shows their related information. It is worthy to note that several papers do not explicitly provide the used dataset, while others use an anonymous dataset for privacy issues. Thus, this article presents only the papers providing open datasets. The Table IV provides the dataset name or the paper reference using it. All the programs are available through quick Google searching.

This study depicts if provided by the paper's authors, the features extracted from the dataset used to train their models that has the potential to predict or classify the alerts. The number of Lines Of Code (LOC) for each program to better know the used dataset's size is also depicted. It is very observable that almost all datasets are not labeled, and authors do not share their manual labeling of SAT alert reports. Only two datasets from NIST and OWASP provide guidelines to label SAT alerts.

TABLE III. COMPARISON OF THE IDENTIFIED APPROACHES

Paper/ Criteria	Security Bugs coverity		Categories							Supported Languages	Scalability			Human labeling effort				False alert reduction rate
	Most Bugs coverity	Only Specific kind of bugs	Machine learning	Data Mining	Model Checking	Root causes	Semantics based	Slicing based	Rule based		Software source code level	Analyzer source code level	Report Directly	None	Few	Moderate	High	
[15][2017]		✓	✓							Java	✓						✓	81.3%– 85%
[39][2017]	✓			✓						Java			✓				✓	not specified
[16][2014]		✓	✓	✓						Java	✓		✓			✓		37.33% – 86.79%
[29][2014]		✓			✓					C++	✓					✓		84% – 84%
[31][2015]		✓			✓					C	✓				✓			70% – not specified
[33][2015]	✓				✓					C		✓			✓			76% – 97.3% (precision)
[17][2017]		✓	✓							C			✓	✓				45%
[40][2018]	✓								✓	OOP Langauges	✓						✓	not specified
[19][2019]	✓		✓							C/C++			✓		✓			61% – 80%
[43][2017]	✓					✓	✓			Java	✓						✓	74%
[25][2016]	✓					✓				JavaScript	✓						✓	36%
[20][2018]	✓		✓							Java/C/C++	✓		✓			✓		not specified
[23][2019]	✓		✓							C/C++	✓						✓	66% – 79%
[4][2018]	✓			✓					✓	Php/Java/C/C++	✓					✓		not specified
[45][2018]		✓					✓			C	✓		✓			✓		prioritization approach
[26][2016]	✓					✓				C	✓		✓				✓	42%
[3][2017]		✓	✓							C	✓				✓			80%
[18][2016]	✓		✓							C	✓				✓			not specified
[34][2010]		✓			✓					C		✓	✓			✓		68%
[49][2012]		✓						✓		C	✓				✓			82%–86%
[50][2018]		✓						✓		Java EE	✓		✓		✓			not specified
[21][2017]		✓	✓							C/C++	✓					✓		81%
[27][2018]		✓				✓				C	✓		✓			✓		6%
[30][2010]	✓				✓					C		✓		✓				96% *
[32][2019]	✓				✓					C	✓		✓			✓		90%
[51][2018]	✓		✓							C			✓	✓				not specified
[44][2020]	✓						✓	✓		C		✓			✓			not specified
[22][2018]	✓		✓							C/C++			✓	✓				not specified
[41][2019]		✓							✓	Java EE	✓						✓	100% **
[38][2013]		✓		✓						C			✓		✓			23.2%

* for only microsoft codes.
** for only one false alert type

A. Interesting Dataset Projects and Competitions

This section presents interesting community projects aiming to provide accurate datasets and enhance static analysis verification research.

1) Juliet Dataset: The National Institue of Standards and Technology (NIST) provides the Software Assurance Reference Dataset (SARD) [1]to users, researchers, security assurance developers to evaluate SAT and test their methods. SARD includes a set of well-known security flaws as test cases covering all software development lifecycles. Also, it covers a large variety of vulnerabilities, languages, platforms, and compilers. The dataset fits all user's needs since it includes wild, synthetic, and academic test cases. It is intended to be a broad effort contributed from many sources[2].

Juliet [3], one of the SARD provided datasets, is a collection of test cases dedicated to C, C++, and Java languages. Juliet's first version 1.0 appeared in December 2010, and its last release, 1.3 delivered in October 2017. It contains examples organized under 118 different CWEs for C/C++ and 112 different CWEs for Java. NIST labels through the methods nominations bad and good codes.

2) OWASP Benchmark Project: OWASP Benchmark Project [4] aims to address the difficulties of testing software defects detection tools and study their weakness, strengths, and analysis time. OWASP provides a Java test suite designed to investigate and evaluate the accuracy, coverage, and speed of Software vulnerabilities analysis and detection tools. OWASP benchmark provides the users with test cases covering all kinds of vulnerabilities and a scoring tool to score the SAT-generated alert and compute the True Positive, False Negative, True Negative, and False Positive alerts percentages.

[1]https://samate.nist.gov/index.php/SARD.html
[2]https://samate.nist.gov/index.php/Software_Assurance_Reference_Dataset.html
[3]https://samate.nist.gov/SRD/testsuite.php
[4]https://owasp.org/www-project-benchmark/

3) Competition on Software Verification: The European Joint Conferences on Theory & Practice of Software, ETAPS[5] organizes each year, starting from 2012, an international competition on software verification to boost the invention of new methods, technologies, and tools to improve the software analysis process.

In the training phase, they provide several benchmark programs, each covering a wide range of CWEs weaknesses to SAT developers. Then, the submitted verifiers' tool will be executed in the evaluation phase, and the number of solved instances and runtime is measured. Researchers could find valuable programs to use as a dataset within the ETAPS website and competition results of each year.

B. Papers' Identified Security Bugs

Table V presents the security bugs handled on different papers to enhance SAT precision to detect potential security bugs without increasing the FPA rate. The paper [19] is the only one that considers almost security bugs during the FPA reduction process.

This section answered the research questions RQ4 and RQ5 by presenting the used datasets, the identified bug types, and the relevant projects and competitions.

VII. DISCUSSION: SHORTCOMINGS AND RECOMMENDATIONS

In the review of the identified approaches, this study depicted several shortcomings that decrease the effectiveness of false alert handling methods. We cite mainly:

- almost cited papers use open programs labeled by themselves without providing their alerts labeling datasets. Which will prohibit other researchers from reproducing the papers' proposed method.

[5]https://etaps.org/about/conferences

TABLE IV. SELECTED SAT DATASETS

Dataset Name	Provider	Type	Language	Labeled	Programs Name	LOC	Extracted Features
Open source code	[16]	projects	Java		axiom guava ivy jenkins-core mahout maven-core opennlp poi rav tika	57,650 64,629 64,629 77,157 264,374 32,322 36,151 292,967 30,762 15,037	# of conditional statements # of loop statements # of return statements # of break or continue statements # of exit or assert method invocations # of null expressions # of comparisons with a null value # of null assignments # statements that return a null value
Open Source	[18]	programs	C	–	brutefir-1.0f consolcalculator-1.0 id3-0.15 mp3rename-0.6 irmp3-0.5.3.1 httptunnel-3.3 e2ps-4.34 less-382 bison-2.5 pies-1.2 icecast-server-1.3.12 raptor-1.4.21 dico-2.0 lsh-2.0.4	103 298 512 2,466 3,797 6,174 6,222 23,822 56,361 66,196 68,564 76,378 84,333 110,898	–
Open source	[3]	libraries	C	–	BIND-1 BIND-2 BIND-3 BIND-4 SM-1 SM-2 SM-3 SM-4 SM-5 SM-6 SM-7 FTP-1 FTP-2 FTP-3	–	is the loop condition contains nulls or not is the loop condition contains constants or not is the loop condition contains array accesses or not (is the loop condition contains && or not is loop condition contains an index for a single array is loop condition contains an index for multiple arrays is the loop condition contains an array index outside the loop is an index is initialized before the loop # of exits in the loop the (normalized) size of the loop # of array accesses in the loop # of arithmetic increments in the loop # of pointer increments in the loop is the loop condition prunes the abstract state or not
Industrial application	[20]	Applications	Java/C++/C	–	Not provided	–	name of the codebase where the alert was detected audit determination full path to the file where the alarm occurs line number in the file where the alert occurs name of the CERT rule associated with the alert title of the CERT rule associated with the alert severity field of the CERT rule likelihood field of the CERT rule remediation field of the CERT rule priority field of the CERT rule level field of the CERT rule name of the function where the alert occurs #of lines of code in the function cyclomatic complexity of the function #of significant lines of code in the function cyclomatic complexity of the function # of parameters to the function # of lexical tokens in the function line number where the function definition starts line number where the function definition ends # of alert that occur in this function Filename # of significant lines of code in the file # of functions/methods in the file average significant lines of code in functions in the file average number of tokens in functions in the file # of alerts that occur in the file depth of the file where the alert occurs
Juliet 1.0 – 1.3	NIST	Test cases	C/C++/JAVA	✓	test cases covering 118 C/C++ CWEs and 112 Java CWEs	–	–
Owasp benchmark	Owasp	Test cases	Java	✓	2,371 data points . 1,193 false positive 1,178 true positive error	–	–
Juliet version 1.2	[19]	Test case + Alarms	C	✓	–	–	Name of the tool generating the warning # alerts in the same file Category of the warning # alerts triggered for the same line by any tool # of alerts less than 4 lines away from the triggered alert is the tool triggered an alert for that location
Open Source	[21]	programs	C/C++	–	bitlbee 4.2 nghttp2 1.6.0 mupdf 1.2.337 h2o 1.7.2 xserver 1.14.3 php 5.6.7	68,413 71,387 122,481 517,731 568,964 709,356	–

TABLE V. SOFTWARE CODING SECURITY BUGS

Paper refference	Language	Bug name
[16]	Java	General null dereference Dereferencing of an unchecked null value Dereferencing of a returned null value
[27][17]	C	Buffer overflow
[19]	C	Buffer overflow related issues Integer overflow and underflow Divisions by zero Uninitialized variable Unused variable Pointer issues (e.g.: Null pointer dereference) Misused operators (e.g.: i f (myVar = bu f [i])fg) Issues with function parameter Expression is either always true or always false Memory issues (e.g., Memory leak, Double free)
[3]	C	Format-string vulnerabilities Buffer-overflow
[15]	C	SQL inject flaw

- only a few papers consider the combination of more than one technique to handle alerts.
- almost proposed methods deal with the software source code refinements and analysis, which is not scalable as handling the SAT report directly.
- machine learning-based techniques require labeling efforts to examine and validate the proposed approach, which inhibits several researchers from using ML techniques.
- most of the proposed SAT false alert reduction approaches cover C, C++, and Java languages. However, languages such as Python used extensively with big data are rarely focused on by researchers.

To address these shortcomings, this study recommends focusing more on:

- combining several techniques parallelly or sequentially to get better accuracy and lower FPA rate. Existing studies on methods combination shows promising results [5].
- focusing more on the slicing approach to decompose extensive application on small slices is highly encouraged since SATs are very useful with small programs.
- focusing on the processing of SAT report directly to provide better scalability and testing easiness. The significant size of the alerts report is a suitable dataset to examine through deep learning techniques.
- thinking on labeling SAT alert report using active learning techniques to reduce the human effort [52].

VIII. CONCLUSION

This paper studied the recent efforts dealing with SAT alerts handling. It provides a new categorization of the used techniques as well as a comparison between the proposed methods. Then, it presents the datasets used to test and validate the different approaches along with information about their size, features, and contained bugs. It summarizes the shortcomings of existing approaches and cites recommendations for future research to improve SAT false alerts handling.

As future plans, profoundly investigating the slicing approach of SAT alert reports and their processing using ML-based techniques will help preserve the SAT scalability and benefit from the high classification accuracy of ML-based methods.

REFERENCES

[1] P. Copeland, "Google's innovation factory: Testing, culture, and infrastructure," in *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 11–14.

[2] R. C. Seacord, D. Plakosh, and G. A. Lewis, *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional, 2003.

[3] K. Heo, H. Oh, and K. Yi, "Machine-learning-guided selectively unsound static analysis," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 519–529.

[4] F. Cheirdari and G. Karabatis, "Analyzing false positive source code vulnerabilities using static analysis tools," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4782–4788.

[5] F. Elberzhager, J. Münch, and V. T. N. Nha, "A systematic mapping study on the combination of static and dynamic quality assurance techniques," *Information and Software Technology*, vol. 54, no. 1, pp. 1–15, 2012.

[6] S. Heckman and L. Williams, "A systematic literature review of actionable alert identification techniques for automated static code analysis," *Information and Software Technology*, vol. 53, no. 4, pp. 363–387, 2011.

[7] V. R. L. de Mendonca, C. L. Rodrigues, F. A. A. de MN Soares, and A. M. R. Vincenzi, "Static analysis techniques and tools: A systematic mapping study," *ICSEA*, 2013.

[8] T. Muske and A. Serebrenik, "Survey of approaches for handling static analysis alarms," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2016, pp. 157–166.

[9] B. Aloraini, M. Nagappan, D. M. German, S. Hayashi, and Y. Higo, "An empirical study of security warnings from static application security testing tools," *Journal of Systems and Software*, vol. 158, p. 110427, 2019.

[10] J. Herter, D. Kästner, C. Mallon, and R. Wilhelm, "Benchmarking static code analyzers," *Reliability Engineering & System Safety*, vol. 188, pp. 336–346, 2019.

[11] P. Nunes, I. Medeiros, J. C. Fonseca, N. Neves, M. Correia, and M. Vieira, "Benchmarking static analysis tools for web security," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1159–1175, 2018.

[12] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 470–481.

[13] A. Arusoaie, S. Ciobâca, V. Craciun, D. Gavrilut, and D. Lucanu, "A comparison of open-source static analysis tools for vulnerability detection in c/c++ code," in *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2017, pp. 161–168.

[14] S. Heckman and L. Williams, "A model building process for identifying actionable static analysis alerts," in *2009 International Conference on Software Testing Verification and Validation*. IEEE, 2009, pp. 161–170.

[15] U. Koc, P. Saadatpanah, J. S. Foster, and A. A. Porter, "Learning a classifier for false positive error reports emitted by static code analysis tools," in *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2017, pp. 35–42.

[16] J. Yoon, M. Jin, and Y. Jung, "Reducing false alarms from an industrial-strength static analyzer by svm," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 2. IEEE, 2014, pp. 3–6.

[17] W. Lee, W. Lee, D. Kang, K. Heo, H. Oh, and K. Yi, “Sound non-statistical clustering of static analysis alarms,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 39, no. 4, pp. 1–35, 2017.

[18] K. Heo, H. Oh, and H. Yang, “Learning a variable-clustering strategy for octagon from labeled data generated by a static analysis,” in *International Static Analysis Symposium*. Springer, 2016, pp. 237–256.

[19] A. Ribeiro, P. Meirelles, N. Lago, and F. Kon, “Ranking warnings from multiple source code static analyzers via ensemble learning,” in *Proceedings of the 15th International Symposium on Open Collaboration*, 2019, pp. 1–10.

[20] L. Flynn, W. Snavely, D. Svoboda, N. VanHoudnos, R. Qin, J. Burns, D. Zubrow, R. Stoddard, and G. Marce-Santurio, “Prioritizing alerts from multiple static analysis tools, using classification models,” in *2018 IEEE/ACM 1st International Workshop on Software Qualities and their Dependencies (SQUADE)*. IEEE, 2018, pp. 13–20.

[21] H. Yan, Y. Sui, S. Chen, and J. Xue, “Machine-learning-guided typestate analysis for static use-after-free detection,” in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 42–54.

[22] E. A. Alikhashashneh, R. R. Raje, and J. H. Hill, “Using machine learning techniques to classify and predict static code analysis tool warnings,” in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2018, pp. 1–8.

[23] S. Lee, S. Hong, J. Yi, T. Kim, C.-J. Kim, and S. Yoo, “Classifying false positive static checker alarms in continuous integration using convolutional neural networks,” in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2019, pp. 391–401.

[24] J. J. Rooney and L. N. V. Heuvel, “Root cause analysis for beginners,” *Quality progress*, vol. 37, no. 7, pp. 45–56, 2004.

[25] J. Park, I. Lim, and S. Ryu, “Battles with false positives in static analysis of javascript web applications in the wild,” in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 61–70.

[26] T. Muske and U. P. Khedker, “Cause points analysis for effective handling of alarms,” in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 173–184.

[27] T. Muske, R. Talluri, and A. Serebrenik, “Repositioning of static analysis alarms,” in *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*, 2018, pp. 187–197.

[28] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[29] M. Valdiviezo, C. Cifuentes, and P. Krishnan, “A method for scalable and precise bug finding using program analysis and model checking,” in *Asian Symposium on Programming Languages and Systems*. Springer, 2014, pp. 196–215.

[30] T. Ball, E. Bounimova, R. Kumar, and V. Levin, “Slam2: Static driver verification with under 4% false alarms,” in *Formal Methods in Computer Aided Design*. IEEE, 2010, pp. 35–42.

[31] B. Chimdyalwar, P. Darke, A. Chavda, S. Vaghani, and A. Chauhan, “Eliminating static analysis false positives using loop abstraction and bounded model checking,” in *International Symposium on Formal Methods*. Springer, 2015, pp. 573–576.

[32] T. T. Nguyen, P. Maleehuan, T. Aoki, T. Tomita, and I. Yamada, “Reducing false positives of static analysis for sei cert c coding standard,” in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*. IEEE, 2019, pp. 41–48.

[33] T. Muske and U. P. Khedker, “Efficient elimination of false positives using static analysis,” in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 270–280.

[34] Y. Kim, J. Lee, H. Han, and K.-M. Choe, “Filtering false alarms of buffer overflow analysis using smt solvers,” *Information and Software Technology*, vol. 52, no. 2, pp. 210–219, 2010.

[35] M. Bharati and M. Ramageri, “Data mining techniques and applications,” 2010.

[36] D. J. Hand, “Principles of data mining,” *Drug safety*, vol. 30, no. 7, pp. 621–622, 2007.

[37] S. Dua and X. Du, *Data mining and machine learning in cybersecurity*. CRC press, 2016.

[38] D. Zhang, D. Jin, Y. Xing, H. Zhang, and Y. Gong, “Automatically mining similar warnings and warning combinations,” in *2013 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, 2013, pp. 783–788.

[39] F. Cheirdari and G. Karabatis, “On the verification of software vulnerabilities during static code analysis using data mining techniques,” in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2017, pp. 99–106.

[40] J. Nam, S. Wang, Y. Xi, and L. Tan, “Designing bug detection rules for fewer false alarms,” in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018, pp. 315–316.

[41] J. Yang, L. Tan, J. Peyton, and K. A. Duer, “Towards better utilizing static application security testing,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 51–60.

[42] J. Euzenat, P. Shvaiko *et al.*, *Ontology matching*. Springer, 2007, vol. 18.

[43] X. Zhang, R. Grigore, X. Si, and M. Naik, “Effective interactive resolution of static analysis alarms,” *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–30, 2017.

[44] D. Kästner, L. Mauborgne, S. Wilhelm, and C. Ferdinand, “High-precision sound analysis to find safety and cybersecurity defects,” in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.

[45] H. Wang, M. Zhou, X. Cheng, G. Chen, and M. Gu, “Which defect should be fixed first? semantic prioritization of static analysis report,” in *International Conference on Software Analysis, Testing, and Evolution*. Springer, 2018, pp. 3–19.

[46] M. Weiser, “Program slices: formal, psychological, and practical investigations of an automatic program abstraction method,” *PhD thesis, University of Michigan*, 1979.

[47] B. Korel and J. Laski, “Dynamic slicing of computer programs,” *Journal of Systems and Software*, vol. 13, no. 3, pp. 187–195, 1990.

[48] H. Agrawal and J. R. Horgan, “Dynamic program slicing,” *ACM SIGPlan Notices*, vol. 25, no. 6, pp. 246–256, 1990.

[49] O. Chebaro, N. Kosmatov, A. Giorgetti, and J. Julliand, “Program slicing enhances a verification technique combining static and dynamic analysis,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012, pp. 1284–1291.

[50] J. Thome, L. K. Shar, D. Bianculli, and L. Briand, “Security slicing for auditing common injection vulnerabilities,” *Journal of Systems and Software*, vol. 137, pp. 766–783, 2018.

[51] A. Ribeiro, P. Meirelles, N. Lago, and F. Kon, “Ranking source code static analysis warnings for continuous monitoring of floss repositories,” in *IFIP International Conference on Open Source Systems*. Springer, 2018, pp. 90–101.

[52] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and S. Madden, “Scaling up crowd-sourcing to very large datasets: a case for active learning,” *Proceedings of the VLDB Endowment*, vol. 8, no. 2, pp. 125–136, 2014.