IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 5, 2021, accepted June 29, 2021, date of publication July 5, 2021, date of current version July 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094517

Measuring Software Obfuscation Quality–A Systematic Literature Review

SHOUKI A. EBAD[1], ABDULBASIT A. DAREM[1], (Member, IEEE), AND JEMAL H. ABAWAJY[2], (Senior Member, IEEE)

[1]Department of Computer Science, Northern Border University, Arar 91431, Saudi Arabia
[2]Cybersecurity Research and Innovation Centre, Deakin University, Burwood, VIC 3125, Australia

Corresponding author: Abdulbasit A. Darem (basit.darem@nbu.edu.sa)

This work was supported by the Deputyship for Research and Innovation through the Ministry of Education in Saudi Arabia under Project 1385.

ABSTRACT Software obfuscation techniques are increasingly being used to prevent attackers from exploiting security flaws and launching successful attacks. With research on software obfuscation techniques rapidly growing, many software obfuscation techniques with varying quality and strength have been proposed in the literature. However, the literature on obfuscation techniques has not yet been coherently collated and reviewed. This research paper aims to present an overview of state-of-the-art software obfuscation techniques, focusing on quality and strength. A systematic analysis and synthesis of literature published between 2010 and April 2021 has been performed to identify the common measures to quantify obfuscation and their measures, the publication venue, and the home country of the researchers. We have identified the obfuscation quality attributes, such as potency, resilience, cost, stealth, and similarity, that are the most widely used metrics to evaluate the quality of obfuscation techniques. In addition, different measures have been used to quantify these qualities, such as complexity (to measure potency), human effort (to measure resilience), efficiency (to estimate cost), and multiclass performance metrics, distance measures, and matching method (to quantify similarity). These measures were then categorized into sub-measures. The literature lacks research in the following two areas: empirical research using a case study strategy, i.e., real-world datasets, and measurements of obfuscation stealth. Researchers did not address stealth as clearly as they addressed potency, cost, and similarity.

INDEX TERMS Software obfuscation, obfuscation quality, obfuscation measure, malware detection, systematic literature review.

I. INTRODUCTION

Software obfuscation is a technique that obscures the structure and/or behavior of software code without impacting its expected functionality such that the code is rendered hard to understand, analyze, or reverse engineer [1], [2]. Software obfuscation techniques are developed for both malicious (e.g., evading automatic static code inspection) and benign (e.g., protecting code privacy or intellectual property) purposes [3]. For example, Malware authors use software obfuscation to evade detection and thwart inspection and removal. Although there is no guarantee that the obfuscated code will be completely immune to reverse engineering, obfuscation increases the effort or cost required to learn the obfuscated code's functionality [1], [4].

The associate editor coordinating the review of this manuscript and approving it for publication was Chunsheng Zhu.

Various obfuscation methods and techniques have been proposed in the exiting works, ranging from syntactic techniques (e.g., inserting opaque predicates) to semantics-based techniques (e.g., complicating control flows). These obfuscation techniques differ in quality and strength. Despite this, there is a lack of measurement in this field to report the quality and strength of the proposed techniques [4], and it remains unclear how ''good'' these techniques are [5]. Measuring the quality of obfuscation techniques is not only useful but also necessary because the obfuscator cannot tell whether the obfuscation is effective if there is no measure of its efficacy. In addition, the developers can improve obfuscation quality by increasing the number and type of obfuscation techniques they use. Therefore, measurement is required for at least evaluating and controlling the obfuscation because, as DeMarco's rule states, ''you can neither predict nor control what you cannot measure'' [6]. In such a context, measurement is

concerned with quantifying an attribute of an obfuscated code. It is nothing more than a number or symbol that is assigned to the obfuscated code to characterize its quality [1]. From the literature, the most common obfuscation quality attributes are cost, resilience, potency, stealth, and similarity. Although such attributes go back to the 1990s, researchers have shown a genuine interest in this area and become competent to discuss obfuscation quality over the last ten years. This interest stems from the fact that the amount of malicious software has increased over the last few years, as approximately 1 million malwares are created daily [7]. This issue has become worse with the adoption of pervasive computing and as technology has shifted toward an Internet of Things.

This review article focuses on the measures used to quantify obfuscation quality in the past decade. To this end, we conduct this systematic literature review (SLR) after analyzing exhaustively relevant studies. The paper makes the following main contributions:

- We perform this SLR according to the key features of measuring software obfuscation quality.
- We provide a systematic overview of the current studies on the area under study.
- We divide the quality attributes of software obfuscation into five categories: potency, resilience, cost, stealth, and similarity. Thereafter, we classify the exiting measures used to quantify the obfuscation quality into 44 sub-measures.
- We make the discussions and provide directions for further studies in measuring obfuscation strength.

The rest of the paper is structured as follows. Section 2 briefly describes the existing work, and the systematic review itself is presented in Section 3. The findings are discussed in Section 4. Section 5 discusses some threats to the study's validity. We conclude the paper in Section 6.

II. EXISTING SYSTEMATIC REVIEWS

Although useful, existing work has not put significant focus on obfuscation quality and strength measurement. In the following, we will discuss several review articles on obfuscation transformation techniques.

A research by Humayun *et al.* [8] was conducted to evaluate the most common cyber security threats based on 78 primary studies. The results showed that as the current security approaches target security generally, more empirical validation and actual implementation is needed for the solutions presented in these studies. Additionally, their findings revealed that most of the research focused on a small number of ordinary flaws, for instance, social engineering, denial-of-service attacks, and malware. Novak *et al.* [9] presented an extensive SLR in academia to detect the similarity of source-code. They looked at 150 primary studies from various angles, including automated tools for detection, performance metrics, methods for obfuscation, deployed datasets, and the types of algorithm used. The multiclass metrics were the most common metrics for assessing system quality and comparing similarity. Khoshavi *et al.* [10] conducted a survey in stack cache memory modules to present a taxonomy of attack vectors, most of which were side-channel attack points. The authors discussed side-channel attacks, including how they work and how obfuscation techniques can help to prevent them. The findings included novel variations of side-channel attacks to perpetrate attacks on systems, as well as countermeasures against the attacks. Asadoorian *et al.* [11] outlined the best practices for security throughout each development stage in the software development life cycle (SDLC). He suggested using the following two types of obfuscation techniques in the coding stage: branch insertion/opaque predicates and obfuscation with random numbers. Hataba and El-Mahdy [12] briefly explained the well-known obfuscation techniques that use various transformations (e.g., layout, control-flow, data, and preventive transformations) and how to implement them. Additionally, they discussed how to evaluate these techniques using a set of reasonable criteria, including potency, resilience, and cost.

Hosseinzadeh *et al.* [4] conducted an SLR of diversification/obfuscation techniques. They collected 357 relevant articles that had been published between 1993 and 2017. There are several techniques to obfuscate a piece of code, each of which is used at various stages of the SDLC and targets different sections of the code. Research gaps included the following: (a) different execution environments still need to benefit from obfuscation techniques; and (b) measurements of the effectiveness of obfuscation. Pan *et al.* [13] performed an SLR to review Android malware detection using a static analysis. They gathered 98 studies published from 2014 to 2020. The static analysis categories included Android characteristics, opcodes, code graphs, and symbolic execution. They concluded that, in terms of detection, the neural network approach outperformed the non-neural network approach. Furthermore, there is still a need to improve identification through the use of novel techniques and the establishment of a single framework for performance evaluation. In the study by Wang *et al.* [14], a SLR was used to identify malicious apps by analyzing the behaviors of apps using different features. The authors examined the extracted features, the feature subset selection techniques, the detection approaches, and the scale of performance evaluation. With a wide use of code obfuscation, extracting useful features from the code using a static analysis is quite difficult. In contrast, when it comes to feature extraction of malware apps, a dynamic analysis outperforms a static analysis. Lu [15] reviewed 14 cybersecurity papers published between 2008 and 2017. The authors divided the articles into individual, employee, and organizational categories. These authors painted a picture of the state of cybersecurity and, using the R Project, created an integrative framework that could potentially text-mine end-users' security behaviors and decision-making processes in the event of a security breach. Balakrishnan and Schulze [16] reviewed code obfuscation techniques that could be applied at the following different abstraction levels: data abstraction, procedural abstraction, data types, and control-flow.

Following the discussion above, measuring software obfuscation quality have not attracted the attention of sufficient researchers. Our work aims to bridge the gap by providing researchers with an overview of the existing obfuscation quality measures that are commonly used to quantify the quality of obfuscation techniques.

III. RESEARCH METHODOLOGY

Recently, the term ''systematic literature review'' has appeared in the title of various information security research papers. It is used in parallel with terms, like ''systematic mapping study'' and ''systematic review'' (for examples, see [4], [8], [9]). A systematic literature review is used for evaluating, identifying, synthesizing, and interpreting a specific research question or subject [17]. As a result, a systematic literature review can identify any gaps in the existing works and suggests areas for further research. In our study, we adopted the suggested guidelines in order to carry out a systematic analysis in [17]–[19]. The study is unbiased and repeatable by other researchers since we follow specific guidelines/protocols [20], which differentiates the systematic review studies from other types of reviews.

The protocol of systematic literature review has five stages. Figure 1 depicts the five steps that were followed here.

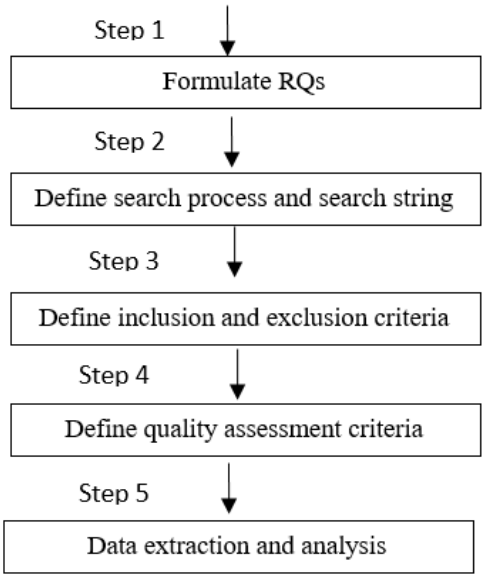

FIGURE 1. Stages of our systematic literature review.

A. QUESTION FORMULATION

To narrow the research target, our systematic review addresses the six key research questions (RQs). Table 1 displays the (RQs) along with their motivations.

B. SEARCH STRATEGY

The keywords used to answer the above RQs were as follows: software, code, program, obfuscation, obfuscate, obfuscator, measurement, measure, metrics, and metric. By taking these keywords and combining them with the ''AND'' and ''OR'' logical connectors, we used the following search command to find relevant articles from the academic libraries under consideration:

(software OR program OR code) AND
(obfuscation OR obfuscate OR obfuscator) AND
(measurement OR measure OR metric OR metrics)

TABLE 1. RQs with their motivations.

ID	Description	Motivation
RQ1	What are the venues for research on measuring the obfuscation strength?	To identify the venues for publications on measurements of obfuscation quality.
RQ2	Which countries are actively measuring the obfuscation quality?	To identify active countries in the research of software security.
RQ3	How do the selected studies conduct their empirical research?	To identify the research methodology and design of the empirical study.
RQ3.1	What environments are the obfuscation quality attributes measured in?	To identify the dataset type of the source that the empirical studies used.
RQ4	What are the common quality attributes of obfuscation?	To identify the key quality attributes based on their frequencies in the primary studies. This helps in understanding the quality attributes relevant to obfuscation and find the hot topics in the field.
RQ5	How are the obfuscation quality attributes measured?	To identify the terms used to quantify obfuscation quality. This answer will help to identify the measures (and their sub-measures, if any) used to quantify the quality attributes in the answer to RQ4. Moreover, the answer finds the granularity level of the used measures.
RQ6	What are the possible directions for future research?	To identify the research gaps in measuring software obfuscation quality

C. SOURCE SELECTION

By using these strings, we can find studies on obfuscation measurement. The above search strings did not aim to search for certain measures applied at a specific level.

The five sources with which our systematic literature review was carried out are SpringerLink, IEEE Explore, ScienceDirect, Wiley Online Library and ACM Digital Library. All the sources include the important journals and conferences in the field, in which significant proposals for obfuscation metrics are presented. It is likely that the primary studies would then include the number of works presented in these sources.

D. STUDIES SELECTION

Our systematic literature review procedure will be iterative and incremental. It is iterative because the systematic review execution is conducted on one search source first, and then on another one. It is incremental; the systematic review document grows with each iteration until it becomes the definitive one, so that its implementation originates from an

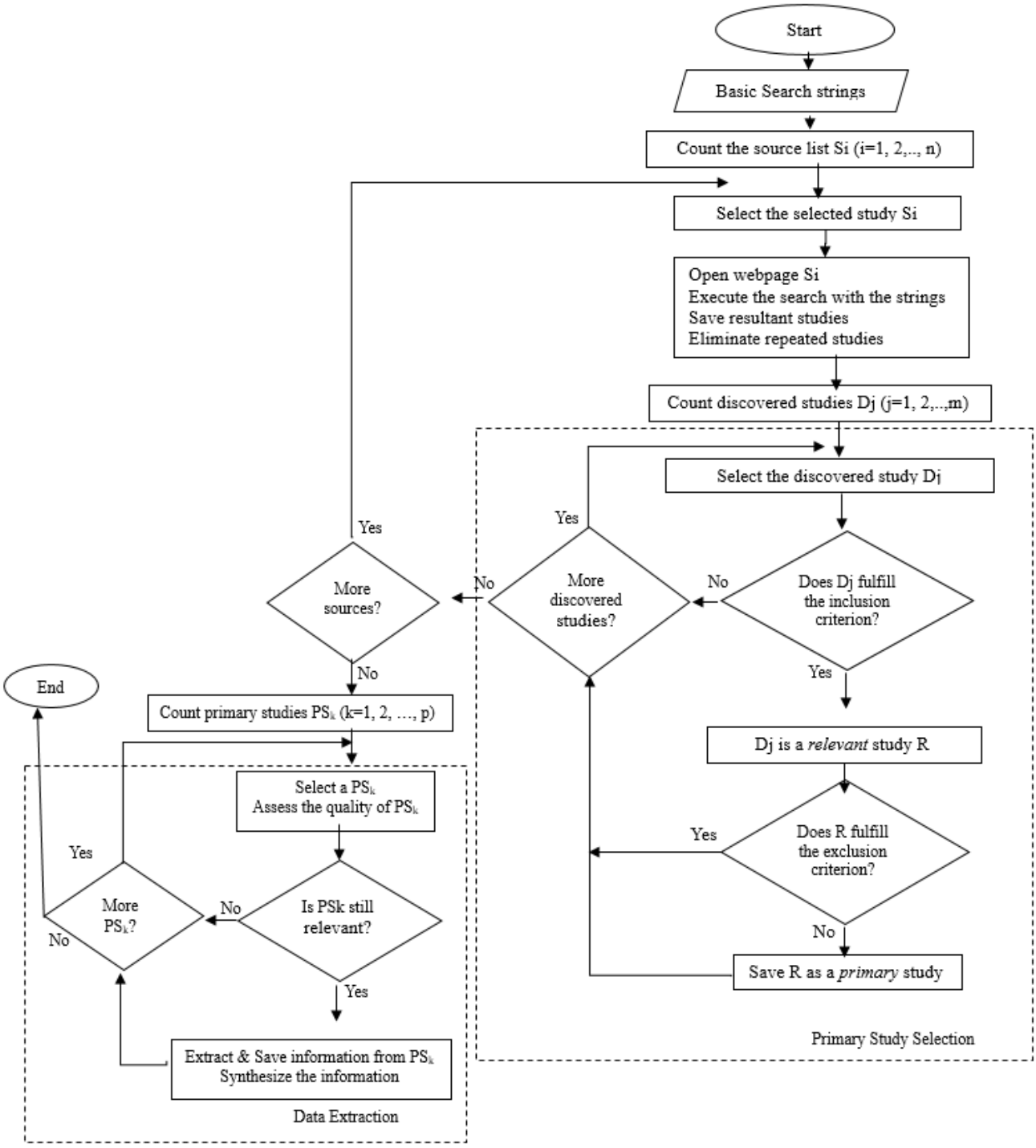

FIGURE 2. The execution of the systematic literature review.

initial source to more sources until the entire review has been performed. In the inclusion criterion, we included studies based on review of the title, abstract, and keywords from the articles acquired in the search. Moreover, the systematic review included studies that were written in English and published between 2010 and April 2021. According to [8], most cyber security threats and crimes were reported after 2007. We applied the following exclusion criteria to the considered abstracts/titles:

- To exclude any paper that does not present an empirical study. we classified the empirical studies to three research methodologies, experiment, case study, and simulation. Simulation methodology was mostly used in the primary studies for validation.

TABLE 2. Distribution of studies after applying the inclusion and exclusion criteria (2010–April 2021).

Library	Discovered studies based on search strings	Repeated and irrelevant studies based on reading Title, Abstract, and Keywords	Exclusive studies	Primary studies
ACM Digital Library	116	73	14	**29** studies ([22][5][23][24][25][3][26][27][28][29][30] [31][32][33][34][35][36][37][38][39][40] [41][42][43][44][45][46][7][47])
IEEE Digital Library	98	39	30	**29** studies (47][49][50][51][52][53][54][55][56][57][58] [59][60][61][62][63][64][65][66][67][43][68] [69][46][70][71][72][73][74])
ScienceDirect	12	7	0	**5** studies ([75][76][77][78][79])
Wiley Online	6	4	1	**1** study ([80])
SpringerLink	70	63	4	**3** studies ([81][82][83])
Total	**302**	**186**	**49**	**67**

- To exclude any paper that does not discuss measuring the obfuscation quality, as this is our area of study.
- To exclude MS/PhD theses, posters, technical reports, and short papers (with less than three pages).
- To exclude the repeated studies.

If we were unsure about a paper after reading its title and abstract, we read the whole the paper. The procedure for retrieving data for the selected studies is shown in Figure 2. Using this procedure, 302 studies were found; 67 of those were selected as primary studies based on the inclusion and exclusion requirements. The specifics of each iteration are shown in Table 2, and possible references within the chosen studies were not sought for.

E. DATA EXTRACTION

As shown in Figure 2, once the relevant studies were selected, the needed information was extracted. An information extraction form (available in Appendix A) was designed to retrieve information from the primary studies. It was reviewed and agreed by all three authors. Besides the data about the selected paper (e.g., title, authors, publication year, and publication country), the form recorded data on the quality assessment of the article such the two main types of threats to the validity: internal validity and external validity.[1] Each threat was scored ''yes'' or ''no'' depending on whether the study explicitly explored the possibility of threats (internal or external validity). Most of the primary studies (59 out of 67) did not mention the internal and external validity, i.e., did not score ''yes'' for both attributes.

F. QUALITY ASSESSMENT

We used the strategy described in Kitchenham and Brereton [17], Kitchenham *et al.* [19] to evaluate the quality of the papers. The same approach has also been used in several SLR studies [98]. Specifically, to ensure that the studies included contribute significantly to the SLR, we developed a checklist criterion based on the guideline proposed in Kitchenham and Brereton [17], Kitchenham *et al.* [19]. The checklist criterion consists of five quality appraisal questions for judging the quality of the studies as shown in the table (Appendix A). Besides the information extraction form (available in the appendix) was reviewed and agreed by all three authors, we assessed the quality of each primary study by means of the attributes: internal validity and external validity. Each was scored 'yes' or 'no' depending upon whether the study explored explicitly the possibility of threats (internal or external validity). In our study, most of the primary studies (59 out of 67) did not score 'yes' for both threats. This was clearly mentioned in the previous version.

IV. RESULTS AND DISCUSSION

First, we needed to find the most recent research trends in the field of measuring obfuscation strength in the security community. The studies distribution by year is shown in Figure 3.

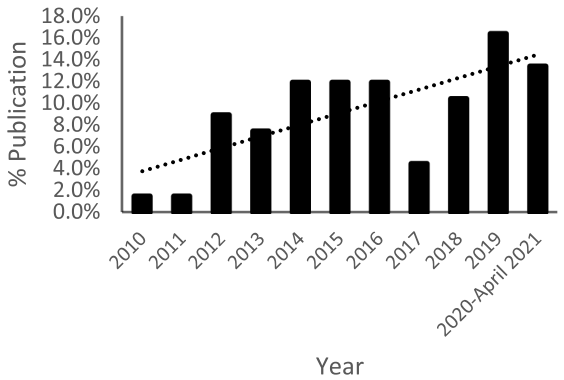

FIGURE 3. Publication trends for the measurement of obfuscation quality.

There is a notable increasing number of studies that deal with the subject. It shows that the number of papers related to this area increased after 2010 (when most cyber issues were reported [8]). The number of studies in 2019 takes the largest proportion. The little downward trends in 2020 comes from

[1]The internal validity is the extent that the independent variable affects causality. The external validity is a condition that limits the ability to generalize the study's results [21].

the fact that not all empirical studies conducted in 2020 were published during this review article.

A. ANALYSIS OF THE PUBLICATION VENUE OF (RQ1)

The first aspect of this article focuses on addressing RQ1, i.e., the source place of selected publications that take an active part in the area of obfuscation measurement. For this analysis, we chose five digital libraries as the main venues, as described in Table 2. The selected studies were published in three types of publications, namely conferences, journals, and workshops/symposiums. Table 3 presents the distribution of the primary articles based on the source type. The proportion of research that have been conducted in conferences and journals was (c. 49, 73%), while (c. 18, 27%) was published in both workshops and symposiums.

The results in Table 3 show that (c. 58, 87%) of primary papers were retrieved from IEEE and ACM libraries. The number of papers in both libraries is almost the same (c. 29, 43%). In contrast, the IEEE library contains more conference papers than the ACM library (17 versus 8). In addition, all workshop papers and symposium papers were extracted from ACM and IEEE libraries except one paper from Springer. Like in previous systematic review studies [8], [20], the contribution of the three other libraries -Springer, Wiley Online, and ScienceDirect- is smaller than that of IEEE and ACM libraries. In these three libraries, the primary studies relevant to the obfuscation measurement were almost published in journals. Additionally, the frequency of papers in ScienceDirect was the highest, with (c. 5, 7%) of the studies. Springer and Wiley Online scored the second and third place with (c. 3, 4%) and (1, 1%), respectively.

TABLE 3. Statistics based on the venue and source types.

Library	Journal	Conference	Workshop/ Symposium	Total
ACM Digital Library	9	8	12	29
IEEE Digital Library	7	17	5	29
ScienceDirect	5	0	0	5
Wiley Online	1	0	0	1
Springer	2	0	1	3
Total	**24**	**25**	**18**	**67**

From Table 3, most the articles retrieved from IEEE and ACM libraries are conference publications. On the other hand, all the primary studies extracted from the other three libraries are almost journal papers. This emphasizes that the main venues for conference papers in this research are IEEE and ACM. Table 4 identifies the key journals and conferences/workshops that publish papers on measuring obfuscation quality. It shows the most used repository with two or more frequencies. The results show that two venues, namely Journal of Computers & Security and the International Workshop on Software Protection, have published more publications in the present domain area of obfuscation measurement with a frequency of six and three, respectively.

TABLE 4. Venues with two studies or more.

Publication Venue	Source Type	Freq.
Computers & Security	Journal	6
International Workshop on Software Protection	Workshop	3
IEEE Access	Journal	2
Journal of Computer Virology and Hacking Techniques	Journal	2
Cybercrime and Trustworthy Computing Conference	Conference	2
International Conference on Dependable Systems and Networks	Conference	2

B. ANALYSIS BASED ON ACTIVE COUNTRIES (RQ2)

We used the author's affiliation to rank the active countries regarding research on the measurement of obfuscation quality, i.e., to answer RQ2. The author's affiliation was used. In case of more than one author, the first author's country was selected. Figure 4 shows the ten authors' affiliation countries, as described in the primary studies. The results (for RQ2) indicate that the four most relevant countries for the primary studies, United States of America (USA), Germany, South Korea, and Singapore contributed (c. 9, 13%, 8, 12%, and 6, 9%, and 5, 7%, respectively). They were followed by authors from China, Australia, and Iran, who contributed (c. 4, 6%) each. Japanese and Indian researchers came after that with a (c. 3, 4%) share. The remaining studies were conducted in different countries with a frequency of between one and two studies, as shown in Figure 4. In contrast, as a continent, Asia leads the statistics with a (c. 30, 45%) share, followed by the United Kingdom (UK) and Europe, and USA and Canada, who contributed (c. 19, 28%) and (c. 10, 15%), respectively. Authors from Africa and Australia were the least associated continents for the selected articles with a (c. 4, 6%) share each. These results are shown in Figure 5. Over 60% of all affiliations are accounted by seven countries, therefore, the research is focused on a specific number of regions. This illustrates the need for more research on software obfuscation from different countries to investigate the effect of sociocultural differences.

TABLE 5. Study strategy used.

Study type	Freq.	%
Experimentation	51	75
Case study	1	1
Simulation	16	24

C. ANALYSIS BASED ON EMPIRICAL STRATEGY AND DATASET (RQ3 & RQ3.1)

As the study is focused on empirical studies, the studies that conducted an empirical validation of the results were chosen. The results in Table 5 (for RQ3) show the distribution of articles based on the used research strategy and that experimentation was the most common strategy with (c. 51, 75%) of the studies using this strategy. The second

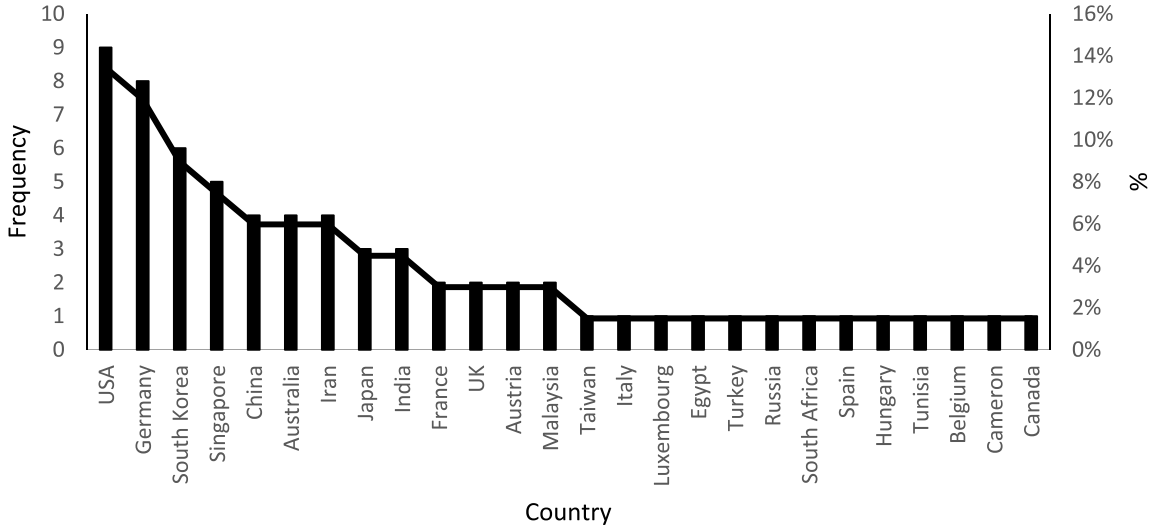

FIGURE 4. Country of publication.

TABLE 6. Study strategy used.

Dataset source category	Freq.	%	Example (study number)
In-the-wild	33	49	Google Play ([35][46][55][58][76][77]), VX Heavens ([3][32][60][65]), F-Droid ([34][56][70][75]), VirusTotal ([36][45][57]), Next Generation Virus Creation Kit ([59][64][66]), Android Malware Dataset ([74]77][81]), MS Windows ([65][66]), Zeus (43][43]), Cygwin ([27]), known worms & viruses ([37])
Manually written programs	10	15	Prime numbers ([23][39][44]), matrix multiplication ([26][44]), bubble sort ([39]), max flow, convex hull, greatest common divisor ([44]), media files ([52]), searching ([54]), Fibonacci ([72])
Open-source software	9	13	Car Race Game ([5][31][33]), Chat ([33])
Benchmark suite	8	12	GNU coreutils ([22]), SPECint 2006 ([24]), SPREE ([51]), DaCapo 9.12 ([61]), National Software Reference Library ([67]), communication protocols- TCP-Modbus & a text-protocol HTTP ([73]), 9.12-bach release of the DaCapo ([78])
Historical data/previous projects	5	7	Data from 200 Windows and Linux hosts ([63]), DROIDCat dataset from Rashidi et al. [86] ([68]), dataset from Sami et al. [87] (69]), dataset from Allix, et al. [88]([46]), dataset from Roy et al. [89]([77])
Web pages	3	4	Cards/applets ([41])

most common methodology was simulation with (c. 16, 24%) of studies, and the least common strategy was a case study, with (c. 1, 1%) share. The total frequency is 68 studies because two different strategies, namely experimentation and simulation, were used in a single study [63]. Table 5 shows that all papers used experimentation or simulation methodologies except for one paper [27] that used a case study methodology. An explanation for the lack of case studies might be that the researcher cannot make meaningful generalizations from this methodology type because it is believed that it does not provide sufficient data to allow generalizability. Moreover, working with a case study from the real-world industry is a popular issue in internet technology research [84] due to confidentiality issues, as details about the participant's organization may be published, i.e., the obfuscation techniques used by their organization. Therefore, this methodology type would generally be high in external validity [21]. Table 6 shows the main sources of the dataset that have been adopted by researchers (RQ3.1). Column one of Table 6 lists the dataset source categories that were identified, column two and three show the frequency and percentage of as the dataset source category appeared in the

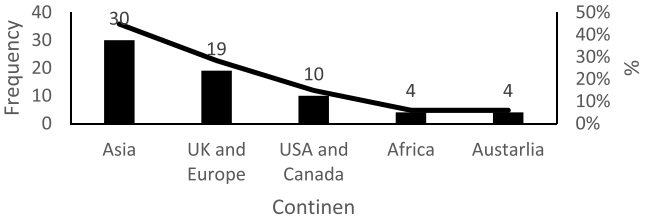

FIGURE 5. Continent of publication.

primary articles, respectively. Herein, the key sources that have been identified are six, in-the-wild, manually written programs, open-source software, benchmark suite, historical data/previous projects, and web pages. The most-widely used dataset source is in-the-wild with a (c. 33, 49%) share, followed by manually written programs, open-source software, benchmark suites, and historical data/previous projects, with (c. 10, 15%, 9, 13%, 8, 12%, and 5, 7%, respectively). The web page-based dataset source was the least widely used source in the primary studies of this systematic literature review with a (c. 3, 4%) share. The last column lists examples of the corresponding source category with the study number. Again, the total frequency is 68 studies because two different

dataset sources, in-the-wild and historical data/previous projects, were used in a single study [77]. The dataset ''in-the-wild'' was the most commonly used dataset in the selected studies. It is continuously updated and maintained [85]; it contains many samples and diverse categories. Google Play Store,[2] VX Heavens,[3] F-Droid,[4] and VirusTotal[5] are examples of such a dataset. Most open-source software was downloaded from SourceForge[6] and GitHub[7] repositories. Most manually written programs that were used as a dataset were written using C language [22], [23], [24], [26], [28]–[30], [44], [47]–[49], [50]. C, Java, and Python are the most common programming languages. The program to generate prime numbers was used in three studies [23], [39], [44].

D. CATEGORIZATION OF QUALITY ATTRIBUTES OF OBFUSCATION (RQ4)

Table 7 shows the main categories of measurements of obfuscation strength (RQ4) that were found from our review article. Column one of Table 7 lists the quality attributes that were found in our study. Column two defines the attribute and its key features, column three shows the corresponding primary study number, and columns four and five show the frequency and percentage of attribute. Compare to the primary studies, the total frequency is more (i.e., 67) because one primary study measured more than one single quality attribute. The key quality attributes that were identified in our study are as follows: potency, resilience, cost, stealth, and similarity. The most widely used obfuscation attribute in the systematic literature review is similarity c. 36, 54%, followed by cost and potency. The number of studies that measured these two attributes is almost the same (c. 21, 31% and 20, 30%, respectively). The least widely used quality attributes in our systematic literature review are resilience and stealth (c. 9, 13% and 8, 12%, respectively).

Although software obfuscation was first developed at the end of 1990 by Collberg *et al.* [1], [90], potency, resilience, cost, and stealth remain the most used obfuscation quality attributes, despite the growing number of security breaches in the last decade.

Another observation is that over half of the primary studies focus on similar attributes. The reason for this high number might come from a serious concern regarding source code plagiarism in academia, i.e., to prevent any act of copying a student's source code with no formal approval. At least once, 72.5 percent of university students confessed to cheating [91]. Several worldwide academic institutes then developed source code similarity detection tools, such as Stanford University in USA, Karlsruhe Institute of Technology in Germany, the University of Sydney in Australia, and the Vrije Universiteit Amsterdam in Netherlands, which developed MOSS,[8] JPlag,[9] Sherlock,[10] and SIM,[11] respectively. Several authors relied on such tools to identify malware programs from benign programs or the malware variants from known variants [30], [42], [7], [47], [43]. Most of the similarity approaches in these studies treat the program as a sequence of bytes. They typically analyze the source code structure, such as the Control Flow Graph (CFG), and contrast it to another source code to find similarities between the two codes. Although existing similarity-based studies generally operate on text strings [59], [82], some studies used more sophisticated methods to handle tokens instead of text [36], [57], [82]. Meanwhile, other similarity approaches tried to detect semantic similarity in source code using a dependency graph [3], [60]. Others attempted to eliminate a large portion of the code, which is less relevant to a similarity comparison [30]. However, complex obfuscation techniques remain difficult to handle [11]. The existing similarity approaches differ in their effectiveness at identifying the obfuscated code generated by obfuscation techniques, which is the similarity score between the non-obfuscated code and its obfuscated/plagiarized version. For this, most of these studies used performance metrics as a sub-measure to estimate the similarity attribute (RQ5 will be answered in the next section).

TABLE 7. Obfuscation quality attributes categorization.

Quality attribute	Brief Overview	Studies	Freq.	%
Potency	What degree is a human reader confused? [1]	[22][5][23][24][25] [31][33][34][35][39] [47][54][55][56][61] [72][73][75][80][81] 78	20	30
Resilience	How well does an obfuscating strategy stand up to automatic deobfuscator attacks? [1]	[28][29][31][40][50] [51][5272][73][7880]	9	13
Cost	How many additional resources are needed to run an obfuscated code? [1]	[24][26][35][38][39] [40][42][47][49][53] [55][56][61][62][66] [68][69][71][72][73 75][8281]	21	31
Stealth	What is the degree to which obfuscated code blends in with non-obfuscated code? [95]	[24][3][27][40][57] [58][71][73]	8	12
Similarity	To what degree do two code samples have the same functionality? [9]	[3][30][32][34][36] [37][39][42][43][44] [45][7][48][51][53] [57][59][60][63][64] [65][67][43][69][46] [70][71][74][76][77] [78][79][81][82][83]	36	54

[2]https://play.google.com/store
[3]https://vx-underground.org/archive/VxHeaven/index.html
[4]https://www.f-droid.org
[5]https://www.virustotal.com/gui
[6]https://sourceforge.net
[7]https://github.com

[8]https://theory.stanford.edu/~aiken/moss
[9]https://jplag.ipd.kit.edu/
[10]https://github.com/diogocabral/sherlock
[11]https://dickgrune.com/Programs/similarity_tester/

TABLE 8. Obfuscation quality measures categorization.

Quality Attribute	Measure	Sub-measure	Studies	Freq.	%
Potency	Complexity	McCabe cyclomatic complexity index	[25][26][33][54][55][72][75]	7	10.4
		Program modularity using OO design metrics	[29][33][61][75]	4	6.0
		Program length	[26][29][73]	3	4.5
		CFG size & depth	[23][75]	2	3.0
		Halstead measure	[54][72]	2	3.0
		Hiding fields accesses	[22]	1	1.5
		Solving comprehension tasks within limited time	[3]	1	1.5
		Errors by obfuscation	[24]	1	1.5
		Nesting complexity	[26]	1	1.5
		Data flow complexity	[26]	1	1.5
		Fan-in/out complexity	[26]	1	1.5
		Data structure complexity	[26]	1	1.5
		Constraints on symbolic variables	[29]	1	1.5
		Number of correctly solved tasks	[31]	1	1.5
		The time spent on correctly solving tasks	[31]	1	1.5
		Number of file open operations	[31]	1	1.5
		Number of executed advanced commands	[31]	1	1.5
		Time of code reading	[31]	1	1.5
		Method reordering	[35]	1	1.5
		Instruction count	[39]	1	1.5
		Branch count	[39]	1	1.5
		Elshoff measure	[72]	1	1.5
Resilience		Programmer experience	[31]	1	1.5
	Human effort	Attacker's time to recover the CFG	[49]	1	1.5
		Attacker's time to recover the hidden data	[49]	1	1.5
		Programmer effort	[72]	1	1.5
		Deobfuscator effort	[72]	1	1.5
		An expert in reverse engineering	[73]	1	1.5
Cost	Efficiency	Time overhead	[24][26][28][38][39][40][4041][56] [61][62][71][72][73][75][81]	16	23.9
		Space overhead	[24][40][40][55][72][73]	6	9.0
		File Size	[55][72]	2	3.0
Similarity	Distance	Entropy (Shannon entropy, behavioral entropy, string entropy, etc.)	[36][39][51][53][64][65]	8	11.9
		Cosine	[37][60][63]	3	4.5
		Longest common subsequence	[44][45][70]	3	4.5
		Edit	[59][67]	2	3.0
		Euclidean	[32][67]	2	3.0
		Comparing fingerprints	[34]	1	1.5
		K-L divergence	[36]	1	1.5
		Hamming distance	[43]	1	1.5
		Gaussian kernel	[46]	1	1.5
		Distance between the n-grams and the context	[74]	1	1.5
	Matching algorithms	Function-name matching	[57][60]	2	3.0
		String matching	[30]	1	1.5
	Multiclass performanc e metrics	Precision, recall, F-measure (f1-score)	[3][43][44][45][46][58][60][62] [65][66][43][68][69][46][71]74] [76][77][81][82]	22	32.8

E. MEASURE-BASED ANALYSIS (RQ5)

The fifth research question (RQ5) was framed to find out the measures of obfuscation quality attributes. As shown in Table 8, the widely used measures to quantify the obfuscation potency, resilience, cost, and similarity are complexity, human effort, efficiency, and multiclass performance metrics, distance measures, and matching method, respectively. Because each measure could be used to control the considered quality attribute at different levels, we have classified them into 44 different sub-measures, as shown in column three.

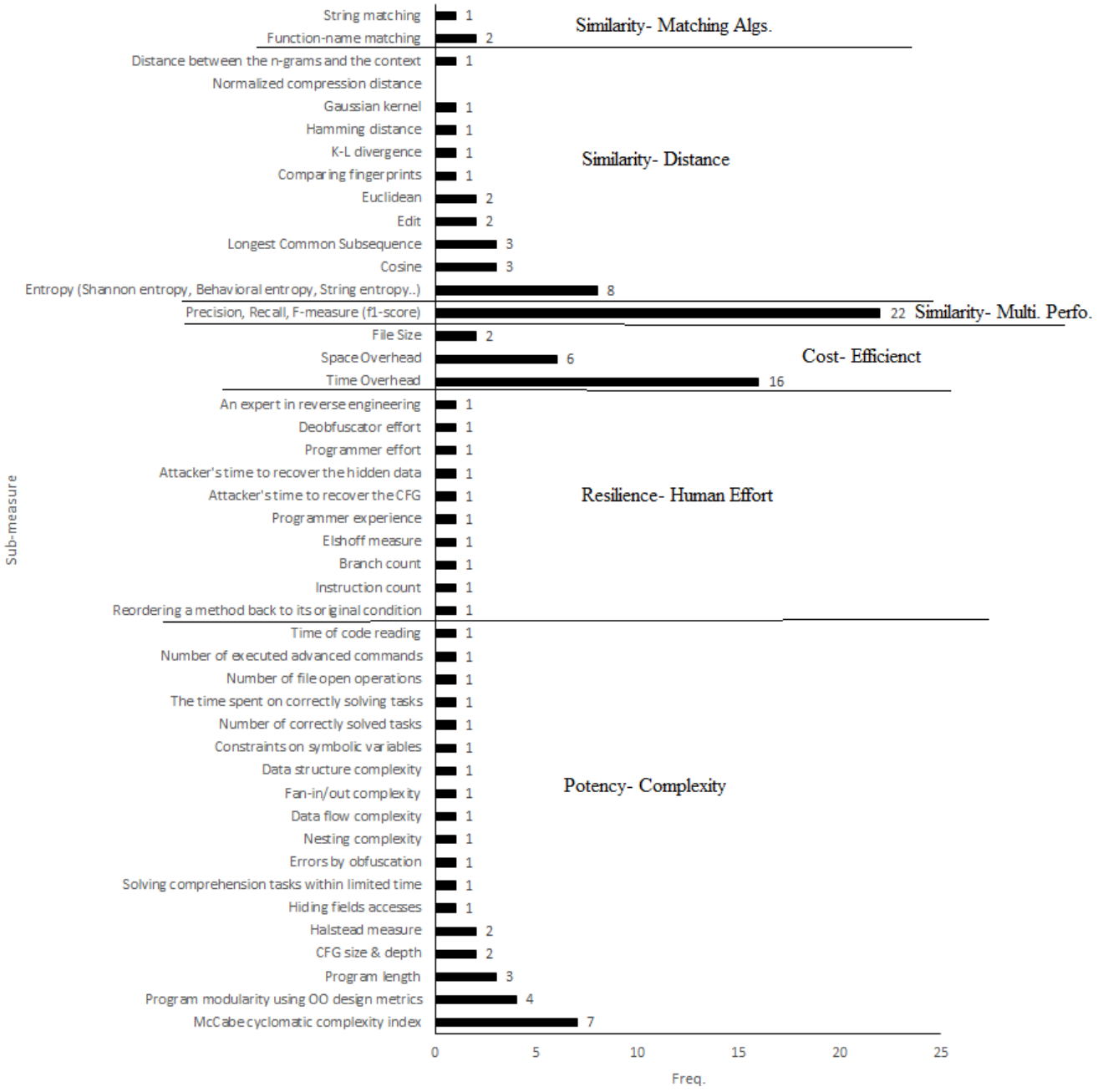

FIGURE 6. **Measures and their sub-measures of obfuscation quality based on their frequency of occurrence.**

Column four lists the primary study numbers, while the frequency and percentage of occurrence for each sub-measure as they appeared in the primary studies are shown in the last two columns (columns five and six). This frequency is illustrated in Figure 6. According to our extracted data, some researchers used more than one measure to quantify a single quality attribute; for example, study [26] used four complexity sub-measures to quantify the potency, and study [49] used two different sub-measures of human efforts to estimate the resilience. Therefore, the total frequency is 84 studies (i.e., greater than the number of primary studies) because of using more than one sub-measure by a single author or study.

According to our literature review, the McCabe cyclomatic number[12] and program object-oriented (OO) metrics[13] are the most common two measures for complexity evaluation, i.e., to quantify the obfuscation potency (c. 7, 10.4% and

[12]The cyclomatic number is computed for the program's CFG, as e-n+2, the CFG has e edges and n nodes. It was defined by McCabe [92]. Although it was originally proposed for procedural programming languages, its adoption in OO languages has often been discussed [25].

[13]The OO metrics were proposed by Chidamber and Kemerer [93].

TABLE 9. Data Extraction Form.

Study data	
Author	Author(s) of the study
Name	Title of the study
Type	Publication types: Journal, Conference, Workshop, or Symposium
Country	The place of the study
Year	The year of the study
Quality assessment	
Does the study clearly state the objectives?	Yes/No
Does the paper clearly describe the methods used?	Yes/No
Are the results clearly stated?	Yes/No
Are the results evaluated empirically?	Yes/No
Internal validity?	Yes/No. The study explicitly explored the possibility of threats to the internal validity.
External validity?	Yes/No. The study explicitly explored the possibility of threats to the external validity.
Section 3: Data extraction	
Methodology	It can be an experiment, case study, or simulation.
Objective	Objectives/aims of the study.
Obfuscation technique	List of the obfuscation techniques used in the study.
Attack type	The type of attack, if any, that was impeded with the help of the obfuscation approach.
Abstraction level	Three abstraction levels of obfuscation: binary machine code, source code, and intermediate representation.
Code granularity	Three granularity levels to apply measures: fine grain (i.e., level of instruction, basic block, loop), medium grain (i.e., level of function/method, and class), and coarse grain (i.e., program, package, subsystem, system).
Context	It includes the dataset and the programming languages used in the study.
Obfuscation category	Two main categories: control flow of a program or its data.
Quality attribute	The obfuscation quality attributes that were supposed to be measured (e.g., potency).
Measure	The metrics used to measure obfuscation quality attributes (e.g., complexity).
Validation	Whether the used measure is validated, either theoretically or empirically.
Data analysis	Whether quantitative or qualitative.
Results	The subjective results of the study.
Performance metrics	It determines the performance metric, if used, such as accuracy, recall, and f-score

4, 6%, respectively). Obfuscation tends to operate on the opposite side of the refactoring principle [94]. While refactoring generally aims to decrease the code complexity and coupling,[14] the obfuscators should propose techniques to increase both metrics (complexity and coupling). For this, different obfuscation techniques take opposite mechanisms with code to make it difficult to analyze. Those techniques then decrease the two metrics, most likely with the ultimate goal of in any other way obstructing interpretation. Class Splitter[15] is an example of such techniques [25].

[14]Coupling is the degree of inter-dependence between modules. In contrast, cohesion is the degree of intra-dependence in a single module. From a software quality perspective, low coupling and high cohesion are two signs of a good design [95].

[15]Class Splitter splits the non-obfuscated classes into obfuscated ones by inserting dummy classes. The rationale for this idea was the class complexity increases with depth of its inheritance tree [1], [96]

Time and space overheads are the two most common measures for efficiency evaluation, i.e., to estimate the obfuscation cost (c. 16, 23.9% and 6, 9%, respectively). In contrast, six sub-measures are used to evaluate human effort, i.e., to quantify the obfuscation resilience (c. 1, 1.5%, each). A common factor among the human effort sub-measure is the effort by both the programmer/reverse engineering expert and attacker. However, the real effort required for reverse engineering is not easy to measure because of the varying experience and skills of the people involved (programmer or attacker). It may take some attackers longer than others to analyze the same code. In the case of similarity attributes, three different sets of measures were used in the evaluation, as follows:

- Distance measures (c. 23, 34.3%). Most of these measures were entropy-based measures (e.g., Shannon

entropy), followed by a cosine measure and longest common subsequence.
- Multiclass performance metrics (c. 22, 32.8%), including precision, recall, and F1-score.
- Matching algorithms (c. 3, 4.5%), such as string matching.

The remaining sub-measures and their frequency are shown in Table 8 and Figure 6. The reader is assumed to be familiar with these measures. Interested readers can consult the relevant studies for further information. Although several of the extracted studies [3], [24], [27], [40], [57], [58], [71], [73] mentioned the stealth quality attribute, the authors were not clear on how to measure it. While one study considered the quality of input to be an indicator of stealth quality [40], two other studies used multiclass performance metrics to measure stealth [58], [71].

The last observation about the measures is their granularity level. There are three granularity levels [95], as follows:

- Fine grain: the measure works at variable and statement levels.
- Medium grain: the measure works at function or method levels.
- Coarse grain: the measure works at the program level.

Herein, the results indicate that most of the measures used to quantify obfuscation quality attributes are medium and coarse-grained, with c. 24, 38% and 26, 41%, respectively. This is practical for large programs that use higher-level structures. Such programs were used as the input for the obfuscation technique in most of the studies; the finely grained measure is unsuitable to quantify attributes in large systems. However, the fine-grained measure may be appropriate when the dataset source is ''manually written programs'' (see Table 6).

F. RESEARCH DIRECTIONS (RQ6)

From this paper, there are gaps in the current literature that need more research. Such research will improve the obfuscation strength and quality. There are four directions for further research (i.e., answering to RQ6):

- Developing a standard measure that can estimate or quantify multiple qualities.
- According to Table 8 and Figure 6, which show the current measures along with their frequency of occurrence, there is a lack of measures to quantify the obfuscation stealth, although there some studies claimed that they measured stealth using very general measures. The researchers were not clear in addressing this issue compared to that with other qualities. A possible reason might be that a stealthy code in one program does not mean that it can be stealthy in another program; stealth is highly context-sensitive.
- There is a need to investigate obfuscation quality through performing case studies from the industry, i.e., using real-world datasets. As shown in Table 5, there is a lack of this type of empirical strategy compared to experimentations and simulations, which use public datasets and open-source code.
- There is little research (3 out of 67 studies [23], [26], [39]) that addresses the obfuscation cost issues when adopting the parallel processing mechanism. The need for this research comes from not using OO languages. In such a case, parallel lines can be drawn with data structures; for instance, measuring data structures used by multiple functions [26].

V. THREATS TO VALIDITY AND LIMITATIONS

Like any research work, there are some limitations that may affect the results. According to [17], the quality assessment of publication in systematic reviews is still an issue. The following are two limitations related to this systematic literature review:

- The extraction process may have resulted in some inaccuracies or bias. Although the chosen databases cover the relevant publication in the obfuscation technique domain, some other studies have been not included because they are published in other databases.
- Missing out some keywords in the search string might be another threat. Although all the relevant keywords have been covered, there is still a possibility that some were missed.

VI. CONCLUSION

In this study, the state-of-the-art measures for quantifying software obfuscation quality according to several attributes were summarized. Systematic literature review has been performed using 67 studies from 2010 to April 2021. After analyzing each publication in detail, we found the following: (1) the contribution of the IEEE and ACM libraries is greater than that of the other libraries in the area of research, and two venues, Journal of Computers & Security and IEEE/ACM International Workshop on Software Protection, were the most common publication venues; (2) researchers from the USA, Germany, South Korea, and Singapore were the most active; (3) the most common empirical strategy was experimentation followed by simulation, and in-the-wild datasets (e.g., Google Play) were the most widely used datasets, followed by manually written programs (e.g. programming assignments at a university, such as prime number generation and matrix multiplication); (4) the five key obfuscation quality attributes that were the most discussed in the primary studies were potency, resilience, cost, stealth, and similarity, and similarity, followed by cost and potency, was the most cited obfuscation quality attributes; and (5) the most widely used measure to quantify the potency, resilience, cost, and similarity were complexity, human effort, efficiency, and the multiclass performance metrics, distance measures, and matching methods, respectively. Because each measure could be understood from different angles, they were classified into 44 sub-measures, as shown in Table 8. The McCabe cyclomatic number and OO metrics are the key sub-measures used to estimate the complexity

(i.e., potency), while time and space overheads were the key sub-measures that were found to quantify the efficiency (i.e., cost). The sub-measures that were used to evaluate the human effort were based on the time spent by the programmer and attacker. Moreover, three different sets of measures were used to evaluate the similarity, distance measures (e.g., entropy-based measures), matching algorithms (e.g., function naming matching) and multiclass performance metrics like F1-score, precision, and recall.

APPENDIX A

See Table 9.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia for funding this research work through Project 1385.

REFERENCES

[1] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997.
[2] M. Madou, "Application security through program obfuscation," Ph.D. dissertation, Dept. Electron. Inf. Syst., Ghent Univ., Ghent, Belgium, 2007.
[3] M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren, and N. Zheng, "A similarity metric method of obfuscated malware using function-call graph," *J. Comput. Virol. Hacking Techn.*, vol. 9, no. 1, pp. 35–47, Feb. 2013.
[4] S. Hosseinzadeh, S. Rauti, S. Laurén, J.-M. Mäkelä, J. Holvitie, S. Hyrynsalmi, and V. Leppänen, "Diversification and obfuscation techniques for software security: A systematic literature review," *Inf. Softw. Technol.*, vol. 104, pp. 72–93, Dec. 2018, doi: 10.1016/j.infsof.2018.07.007.
[5] Y. Zhuang, M. Protsenko, T. Muller, and F. C. Freiling, "An(other) exercise in measuring the strength of source code obfuscation," in *Proc. 25th Int. Workshop Database Expert Syst. Appl.*, Sep. 2014, pp. 1–5.
[6] T. DeMarco, *Controlling Software Projects*. New York, NY, USA: Yourdon Press, 1982.
[7] Ö. Aslan, R. Samet, and Ö. Ö. Tanrıöver, "Using a subtractive center behavioral model to detect malware," *Secur. Commun. Netw.*, vol. 2020, pp. 1–17, Feb. 2020.
[8] M. Humayun, M. Niazi, N. Jhanjhi, M. Alshayeb, and S. Mahmood, "Cyber security threats and vulnerabilities: A systematic mapping study," *Arabian J. Sci. Eng.*, vol. 45, no. 4, pp. 3171–3189, Apr. 2020.
[9] M. Novak, M. Joy, and D. Kermek, "Source-code similarity detection and detection tools used in academia: A systematic review," *ACM Trans. Comput. Educ.*, vol. 19, no. 3, pp. 27:1–27:37, 2019.
[10] N. Khoshavi, M. Maghsoudloo, Y. Bi, W. Francois, L. Jaimes, and A. Sargolzaei, "A survey on attack vectors in stack cache memory," *Integration*, vol. 72, pp. 134–147, May 2020.
[11] A. Asadoorian, M. Alberto, and M. L. Ali, "Creating and using secure software," in *Proc. 11th IEEE Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2020, pp. 28–31, doi: 10.1109/UEMCON51285.2020.9298046.
[12] M. Hataba and A. El-Mahdy, "Cloud protection by obfuscation: Techniques and metrics," in *Proc. 7th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.*, Nov. 2012, pp. 12–14, doi: 10.1109/3PGCIC.2012.18.
[13] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A systematic literature review of Android malware detection using static analysis," *IEEE Access*, vol. 8, pp. 116363–116379, 2020, doi: 10.1109/ACCESS.2020.3002842.
[14] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.
[15] Y. Lu, "Cybersecurity research: A review of current research topics," *J. Ind. Integr. Manage.*, vol. 3, no. 4, Dec. 2018, Art. no. 1850014.
[16] A. Balakrishnan and C. Schulze, "Code obfuscation literature survey," Univ. Wisconsin, Madison, WI, USA, Tech. Rep., 2005.
[17] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2049–2075, Dec. 2013.
[18] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, Aug. 2015.
[19] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "The value of mapping studies—A participant-observer case study," in *Proc. 14th Int. Conf. Eval. Assessment Softw. Eng.*, Keele, U.K.: Keele Univ., Apr. 2010, pp. 12–13.
[20] S. A. Ebad, "Inspection reading techniques applied to software artifacts—A systematic review," *Comput. Syst. Sci. Eng.*, vol. 32, no. 3, pp. 213–226, 2017.
[21] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.
[22] S. Blazy and S. Riaud, "Measuring the robustness of source program obfuscation," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2014, pp. 123–126.
[23] Y.-L. Huang and H.-Y. Tsai, "A framework for quantitative evaluation of parallel control-flow obfuscation," *Comput. Secur.*, vol. 31, no. 8, pp. 886–896, Nov. 2012.
[24] V. Balachandran and S. Emmanuel, "Potent and stealthy control flow obfuscation by stack based self-modifying code," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 4, pp. 669–681, Apr. 2013.
[25] M. Ceccato, A. Capiluppi, P. Falcarin, and C. Boldyreff, "A large study on the effect of code obfuscation on the quality of Java code," *Empirical Softw. Eng.*, vol. 20, no. 6, pp. 1486–1524, Dec. 2015.
[26] B. Bertholon, S. Varrette, and S. Martinez, "ShadObf: A C-Source obfuscator based on multi-objective optimisation algorithms," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, May 2013, pp. 20–24.
[27] Y. Kanzaki, A. Monden, and C. Collberg, "Code artificiality: A metric for the code stealth based on an N-Gram model," in *Proc. IEEE/ACM 1st Int. Workshop Softw. Protection*, May 2015, pp. 31–37.
[28] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner, "Code obfuscation against symbolic execution attacks," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, Dec. 2016, pp. 189–200.
[29] S. Banescu, C. Collberg, and A. Pretschner, "Predicting the resilience of obfuscated code against symbolic execution attacks via machine learning," in *Proc. 26th USENIX Secur. Symp.*, Vancouver, BC, Canada, Aug. 2017, pp. 661–678.
[30] S. Park, S. Ko, J. Choi, H. Han, S.-J. Cho, and J. Choi, "Detecting source code similarity using code abstraction," in *Proc. 7th Int. Conf. Ubiquitous Inf. Manage. Commun. (ICUIMC)*, Jan. 2013, pp. 1–9.
[31] N. Hänsch, A. Schankin, M. Protsenko, F. Freiling, and Z. Benenson, "Programming experience might not help in comprehending obfuscated source code efficiently," in *Proc. 14th Symp. Usable Privacy Secur.*, Baltimore, MD, USA, Aug. 2018, pp. 341–356.
[32] B. Rad and M. Masrom, "Metamorphic virus variants classification using opcode frequency histogram," in *Proc. 14th WSEAS Int. Conf. Comput.*, Corfu Island, Greece, Jul. 2010, pp. 147–155.
[33] A. Capiluppi, P. Falcarin, and C. Boldyreff, "Code defactoring: Evaluating the effectiveness of Java obfuscations," in *Proc. 19th Work. Conf. Reverse Eng.*, Oct. 2012, pp. 71–80.
[34] J. Feichtner and C. Rabensteiner, "Obfuscation-resilient code recognition in Android apps," in *Proc. 14th Int. Conf. Availability, Rel. Secur.*, Aug. 2019, pp. 1–10.
[35] V. Balachandran, Sufatrio, D. J. J. Tan, and V. L. L. Thing, "Control flow obfuscation for Android applications," *Comput. Secur.*, vol. 61, pp. 72–93, Aug. 2016.
[36] J. Su, K. Yoshioka, J. Shikata, and T. Matsumoto, "An efficient method for detecting obfuscated suspicious JavaScript based on text pattern analysis," in *Proc. ACM Int. Workshop Traffic Meas. Cybersecurity*, May 2016, pp. 3–11.
[37] M. K. Shankarapani, S. Ramamoorthy, R.S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *J. Comput. Virology*, vol. 7, no. 2, pp. 107–119, 2011.
[38] S. Han, M. Ryu, J. Cha, and B. U. Choi, "HOTDOL: HTML obfuscation with text distribution to overlapping layers," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, Sep. 2014, pp. 11–13.
[39] R. Omar, A. El-Mahdy, and E. Rohou, "Arbitrary control-flow embedding into multiple threads for obfuscation: A preliminary complexity and performance analysis," in *Proc. 2nd Int. Workshop Secur. Cloud Comput. (SCC)*, 2014, pp. 51–58.
[40] A. Ibrahim and S. Banescu, "StIns4CS: A state inspection tool for C#," in *Proc. ACM Workshop Softw. PROtection*, Oct. 2016, pp. 61–71.
[41] M. Lackner, R. Berlach, R. Weiss, and C. Steger, "Countering type confusion and buffer overflow attacks on Java smart cards by data type sensitive obfuscation," in *Proc. 1st Workshop Cryptogr. Secur. Comput. Syst. (CS2)*, 2014, pp. 19–24.

[42] Y. Park and S. J. Stolfo, "Software decoys for insider threat," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, May 2012, pp. 93–94.
[43] A. Azab, R. Layton, M. Alazab, and J. Oliver, "Mining malware to detect variants," in *Proc. 5th Cybercrime Trustworthy Comput. Conf.*, Auckland, New Zealand, Nov. 2014, pp. 44–532.
[44] F. Zhang, Y.-C. Jhi, D. Wu, P. Liu, and S. Zhu, "A first step towards algorithm plagiarism detection," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, Jul. 2012, pp. 111–121.
[45] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, Jun. 2015, Art. no. 659101.
[46] E. Alfs, D. Caragea, D. Chaulagain, S. Roy, N. Albin, and P. Poggi-Corradini, "Identifying Android malware using network-based approaches," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, Aug. 2019, pp. 903–906.
[47] A. K. Biswas, "Cryptographic software IP protection without compromising performance or timing side-channel leakage," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 2, pp. 1–20, Mar. 2021.
[48] X. Zhang, J. Pang, and X. Liu, "Common program similarity metric method for anti-obfuscation," *IEEE Access*, vol. 6, pp. 47557–47565, 2018.
[49] S. Banescu, M. Ochoa, and A. Pretschner, "A framework for measuring software obfuscation resilience against automated attacks," in *Proc. IEEE/ACM 1st Int. Workshop Softw. Protection*, May 2015, pp. 45–51.
[50] M. Talukder, S. Islam, and P. Falcarin, "Analysis of obfuscated code with program slicing," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)*, Jun. 2019, pp. 3–4.
[51] M. Fyrbiak, S. Rokicki, N. Bissantz, R. Tessier, and C. Paar, "Hybrid obfuscation to protect against disclosure attacks on embedded microprocessors," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 207–321, Mar. 2018.
[52] Y.-C. Jhi, X. Jia, X. Wang, S. Zhu, P. Liu, and D. Wu, "Program characterization using runtime values and its application to software plagiarism detection," *IEEE Trans. Softw. Eng.*, vol. 41, no. 9, pp. 925–943, Sep. 2015.
[53] M. Jodavi, M. Abadi, and E. Parhizkar, "JSObfusDetector: A binary PSO-based one-class classifier ensemble to detect obfuscated JavaScript code," in *Proc. Int. Symp. Artif. Intell. Signal Process. (AISP)*, Mar. 2015, pp. 322–327.
[54] M. Styugin, V. Zolotarev, A. Prokhorov, and R. Gorbil, "New approach to software code diversification in interpreted languages based on the moving target technology," in *Proc. IEEE 10th Int. Conf. Appl. Inf. Commun. Technol. (AICT)*, Oct. 2016, pp. 12–14.
[55] L. Zhang, H. Meng, and V. L. L. Thing, "Progressive control flow obfuscation for Android applications," in *Proc. TENCON IEEE Region Conf.*, Oct. 2018, pp. 1075–1079, doi: 10.1109/TENCON.2018.8650141.
[56] Q. Zeng, L. Luo, Z. Qian, X. Du, Z. Li, C.-T. Huang, and C. Farkas, "Resilient user-side Android application repackaging and tampering detection using cryptographically obfuscated logic bombs," *IEEE Trans. Depend. Sec. Comput.*, early access, Dec. 5, 2019, doi: 10.1109/TDSC.2019.2957787.
[57] P. Wrench and B. Irwin, "Detecting derivative malware samples using deobfuscation-assisted similarity analysis," *SAIEE Afr. Res. J.*, vol. 107, no. 2, pp. 65–77, Jun. 2016.
[58] Z. Qu, S. Alam, Y. Chen, X. Zhou, W. Hong, and R. Riley, "DyDroid: Measuring dynamic code loading and its security implications in Android applications," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 415–426.
[59] S. Rezaei, A. Afraz, F. Rezaei, and M. R. Shamani, "Malware detection using opcodes statistical features," in *Proc. 8th Int. Symp. Telecommun. (IST)*, Sep. 2016, pp. 151–155.
[60] L. Wu, M. Xu, J. Xu, N. Zheng, and H. Zhang, "A novel malware variants detection method based on function-call graph," in *Proc. IEEE Conf. Anthology*, Jan. 2013, pp. 1–8, doi: 10.1109/ANTHOLOGY.2013.6784887.
[61] W. Liu and W. Li, "Unifying the method descriptor in Java obfuscation," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 1397–1401.
[62] M. Mimura and Y. Suga, "Filtering malicious JavaScript code with Doc2 Vec on an imbalanced dataset," in *Proc. 14th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2019, pp. 24–31.
[63] S. Wang and P. S. Yu, "Heterogeneous graph matching networks: Application to unknown malware detection," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 5401–5408.
[64] K. Vahedi, M. Abbaspour, K. Afhamisisi, and M. Rashidnejad, "Behavioral entropy towards detection of metamorphic malwares," in *Proc. 9th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2019, pp. 24–25.
[65] X. Ugarte-Pedrero, I. Santos, B. Sanz, C. Laorden, and P. G. Bringas, "Countering entropy measure attacks on packed software detection," in *Proc. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2012, pp. 164–168.
[66] A. Azhikoden and P. Vinod, "Meta opcode space for morphed malware detection," in *Proc. 11th Int. Conf. Innov. Inf. Technol. (IIT)*, Nov. 2015, pp. 284–289.
[67] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Detecting homoglyph attacks with a siamese neural network," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 22–28.
[68] M. Ghiasi, A. Sami, and Z. Salehi, "Dynamic malware detection using registers values set analysis," in *Proc. 9th Int. ISC Conf. Inf. Secur. Cryptol.*, Sep. 2012, pp. 54–59.
[69] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-evolving Android malware detection system," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 47–62.
[70] B. Kim, K. Lim, S.-J. Cho, and M. Park, "RomaDroid: A robust and efficient technique for detecting Android app clones using a tree structure and components of each App's manifest file," *IEEE Access*, vol. 7, pp. 72182–72196, 2019.
[71] S. Ko, J. Choi, and H. Kim, "COAT: Code obfuscation tool to evaluate the performance of code plagiarism detection tools," in *Proc. Int. Conf. Softw. Secur. Assurance (ICSSA)*, Jul. 2017, pp. 32–37.
[72] D. Dunaev and L. Lengyel, "Cognitive evaluation of intermediate level obfuscator," in *Proc. 5th IEEE Conf. Cognit. Infocommun. (CogInfoCom)*, Nov. 2014, pp. 521–525.
[73] J. Duchene, E. Alata, V. Nicomette, M. Kaaniche, and C. Le Guernic, "Specification-based protocol obfuscation," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2018, pp. 478–489.
[74] M. Yousefi-Azar, L. Hamey, V. Varadharajan, and S. Chen, "Byte2vec: Malware representation and feature selection for Android," *Comput. J.*, vol. 63, no. 8, pp. 1125–1138, Aug. 2020.
[75] Y. Zhuang, "The performance cost of software obfuscation for Android applications," *Comput. Secur.*, vol. 73, pp. 57–72, Mar. 2018.
[76] L. Li, T. Riom, T. F. Bissyandé, H. Wang, J. Klein, and L. T. Yves, "Revisiting the impact of common libraries for Android-related investigations," *J. Syst. Softw.*, vol. 154, pp. 157–175, Aug. 2019.
[77] M. Jerbi, Z. C. Dagdia, S. Bechikh, and L. B. Said, "On the use of artificial malicious patterns for Android malware detection," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101743.
[78] A. Pinhero, A. M L, V. P, C. A. Visaggio, A. N, A. S, and A. S, "Malware detection employed by visualization and deep neural network," *Comput. Secur.*, vol. 105, Jun. 2021, Art. no. 102247.
[79] Y. T. Ling, N. F. M. Sani, M. T. Abdullah, and N. A. W. A. Hamid, "Structural features with nonnegative matrix factorization for metamorphic malware detection," *Comput. Secur.*, vol. 104, May 2021, Art. no. 102216.
[80] C. Foket, K. De Bosschere, and B. De Sutter, "Effective and efficient Java-type obfuscation," *Softw., Pract. Exper.*, vol. 50, no. 2, pp. 136–160, Feb. 2020.
[81] F. Tchakounté, R. C. N. Ngassi, V. C. Kamla, and K. P. Udagepola, "LimonDroid: A system coupling three signature-based schemes for profiling Android malware," *Iran J. Comput. Sci.*, vol. 4, no. 2, pp. 95–114, Jun. 2021, doi: 10.1007/s42044-020-00068-w.
[82] J. Chen, M. H. Alalfi, T. R. Dean, and Y. Zou, "Detecting Android malware using clone detection," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 942–956, Sep. 2015.
[83] K. A. Dhanya, O. K. Dheesha, T. G. Kumar, and P. Vinod, "Detection of obfuscated mobile malware with machine learning and deep learning models," in *Machine Learning and Metaheuristics Algorithms, and Applications* (Communications in Computer and Information Science), vol. 1366, S. M. Thampi, S. Piramuthu, K. C. Li, S. Berretti, M. Wozniak, and D. Singh, Eds. Singapore: Springer, 2021, pp. 221–231, doi: 10.1007/978-981-16-0419-5_18.
[84] S. A. Ebad, "An exploratory study of ICT projects failure in emerging markets," *J. Global Inf. Technol. Manage.*, vol. 21, no. 2, pp. 139–160, Apr. 2018.
[85] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. Le Traon, "Empirical assessment of machine learning-based malware detectors for Android," *Empirical Softw. Eng.*, vol. 21, no. 1, pp. 183–211, Feb. 2016.
[86] B. Rashidi and C. Fung, "XDroid: An Android permission control using hidden Markov chain and online learning," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2016, pp. 46–54.
[87] A. Sami, B. Yadegari, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2010, pp. 1020–1025.

[88] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, May 2016, pp. 14–15.
[89] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara, "Experimental study with real-world data for Android app security analysis using machine learning," in *Proc. 31st Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2015, pp. 81–90.
[90] C. Collberg, C. Thomborson, and D. Low, "Manufacturing cheap, resilient, and stealthy opaque constructs," in *Proc. 25th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang. (POPL)*, 1998, pp. 184–196, doi: 10.1145/268946.268962.
[91] D. Sraka and B. Kaucic, "Source code plagiarism," in *Proc. ITI 31st Int. Conf. Inf. Technol. Interface*, Jun. 2009, pp. 461–466, doi: 10.1109/ITI.2009.5196127.
[92] T. McCabe, "A software complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
[93] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–498, Jun. 1994.
[94] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2018.
[95] S. A. Ebad and M. Ahmed, "Review and evaluation of cohesion and coupling metrics at package and subsystem level," *J. Softw.*, vol. 11, no. 6, pp. 598–605, 2016.
[96] S. Banescu and A. Pretschner, "A tutorial on software obfuscation," *Adv. Comput.*, vol. 108, pp. 283–353, Jan. 2018.
[97] S. Huda, S. Alyahya, M. Mohsin Ali, S. Ahmad, J. Abawajy, H. Al-Dossari, and J. Yearwood, "A framework for software defect prediction and metric selection," *IEEE Access*, vol. 6, pp. 2844–2858, 2018.
[98] M. K. Najafabadi and M. N. Mahrin, "A systematic literature review on the state of research and practice of collaborative filtering technique and implicit feedback," *Artif. Intell. Rev.*, vol. 45, no. 2, pp. 167–201, 2016.

SHOUKI A. EBAD received the Ph.D. degree in computer science and engineering from the King Fahd University of Petroleum and Minerals, Saudi Arabia, in 2012. He held several positions, such as a lecturer, the head of the department, the vice dean, the Assistant Dean for technical affairs at the IT Deanship, and the Secretary of the Scientific Council. He is currently working as an Associate Professor with the Department of Computer Science, Faculty of Science, Northern Border University, Saudi Arabia. He is also a Sun Certified Programmer of the Java 2 Platform. He has published a number of articles in the following areas. His current research interests include software engineering, software project management, software security, and cyber security.

ABDULBASIT A. DAREM (Member, IEEE) received the Ph.D. degree in computer science from the University of Mysore, India, in 2014. He has more than 20 years of experience in the IT field. He is currently an Assistant Professor with the Department of Computer Science, Northern Border University, Saudi Arabia. He has published more than 19 research papers in reputed international journals and conferences. His research interests include cybersecurity, Web engineering, HCI, usability, e-government, and cloud computing.

JEMAL H. ABAWAJY (Senior Member, IEEE) received the B.S.E., M.Sc., Ph.D., and D.Sc. degrees. He is currently a Full Professor with the Faculty of Science, Engineering, and Built Environment, Deakin University, Australia. He has delivered more than 60 keynotes worldwide and has authored/coauthored several books and about 400 refereed articles in premier venues and supervised numerous Ph.D. students to completion. He has been actively involved in the organization of more than 350 conferences all over the world in various capacity, including the chair, the general co-chair, the vice-chair, the best paper award chair, the publication chair, and the session chair and involved in the program committee. He has also served/serving on the Editorial Board for numerous international journals, including the IEEE TRANSACTIONS ON CLOUD COMPUTING.