# Data Preparation for Software Vulnerability Prediction: A Systematic Literature Review

Roland Croft, Yongzheng Xie, and M. Ali Babar

**Abstract**—Software Vulnerability Prediction (SVP) is a data-driven technique for software quality assurance that has recently gained considerable attention in the Software Engineering research community. However, the difficulties of preparing Software Vulnerability (SV) related data remains as the main barrier to industrial adoption. Despite this problem, there have been no systematic efforts to analyse the existing SV data preparation techniques and challenges. Without such insights, we are unable to overcome the challenges and advance this research domain. Hence, we are motivated to conduct a Systematic Literature Review (SLR) of SVP research to synthesize and gain an understanding of the data considerations, challenges and solutions that SVP researchers provide. From our set of primary studies, we identify the main practices for each data preparation step. We then present a taxonomy of 16 key data challenges relating to six themes, which we further map to six categories of solutions. However, solutions are far from complete, and there are several ill-considered issues. We also provide recommendations for future areas of SV data research. Our findings help illuminate the key SV data practices and considerations for SVP researchers and practitioners, as well as inform the validity of the current SVP approaches.

**Index Terms**—Systematic Literature Review, Data Preparation, Data Quality, Software Vulnerability Prediction

## 1 Introduction

Software security is a paramount concern due to the continued increase in cybersecurity attacks and exploits that are affecting organizations [1]. Software security techniques often focus on detecting and preventing Software Vulnerabilities (SVs) that make their way into a software product or deployment pipeline before release [2]. These SVs are a unique class of software defects that introduce security weaknesses to software and allow for malicious use of products [3]. Due to the high importance of removing these security defects, considerable research efforts have been conducted towards their mitigation [4].

Software Vulnerability Prediction (SVP) is a data-driven process for software quality assurance that aims to leverage historical SV knowledge to classify code modules as vulnerable or not. The granularity of the modules can be set as needed, such as file, function, or code snippet. This area of research has recently surged in popularity within the research community [4], due to its importance and value. SVP can ensure software security early in development and solve the incapabilities of manually assessing large-scale software systems for potential SVs, which holds inherent value to an organisation.

Like any data-driven process, data preparation serves as one of the most pivotal components for SVP [5]; *garbage in, garbage out*. Consequently, significant efforts need to be expended for data collection and processing [6]. For SVP, we require examples of both vulnerable and non-vulnerable code for training models. Unfortunately, SV data preparation is not a trivial task [7]. High-quality SV data is notoriously difficult to obtain due to its natural infrequency [8], inconsistent reporting [9], and the unwillingness of organisations to make their sensitive data public [10]. It is widely recognized that data noise can severely impact the quality of an SVP model and eventually negatively impact the validity of the research outcomes [11], [12]. That is why datasets are commonly listed as one of the key challenges for this research area [4], [13]. These data quality issues, in combination with the extreme data collection effort requirements, have led many to view data as the major barrier to industrial adoption of SVP [14], [15].

However, despite the importance and difficulties of SV data preparation for both industry and academia, there has been relatively little effort allocated to systematically understand the known challenges of data preparation for SVP models and how to address them. Whilst there are several secondary studies that have analyzed SVP research [4], [13], [16], [17] and acknowledged the existence of problems with the data preparation, these studies have not focused on thoroughly investigating the data preparation related challenges in SVP research.

Motivated by a lack of an integrated and comprehensive body of knowledge on this important topic, we aim to highlight the state of the practice of data preparation for SVP and consequently identify the associated SV data preparation challenges and solutions. This knowledge is expected to assist practitioners and researchers in gaining better understanding of the data preparation challenges in SVP and available solutions, in order to support the development and application of more reliable and trustworthy SVP models. In this paper, we empirically examine and synthesize the current practices, challenges and solutions for SVP data preparation through a Systematic Literature Review (SLR). We systematically select 61 peer-reviewed

- *R. Croft, Y. Xie, and M. A. Babar are affiliated with the Centre for Research on Engineering Software Technologies (CREST), School of Computer Science, University of Adelaide, Adelaide, Australia.
E-mail: {roland.croft, yongzheng.xie, ali.babar}@adelaide.edu.au*
- *R. Croft and M. A. Babar are affiliated with the Cyber Security Cooperative Research Centre, Australia.*

*Manuscript received...*

papers on SVP research. We communicate the state of the practice for SVP data preparation techniques, the reported data challenges of these primary studies, and the existing solutions that researchers have used to combat these issues. The main contributions of this research are:

- The first, to the best of our knowledge, systematic review aimed at systematically developing an integrated and comprehensive source of information regarding SVP data preparation practices, challenges and solutions,
- A taxonomy of 16 SV data challenges across six themes. This taxonomy can be used to classify data challenges for future SVP research and practice,
- A mapping of the identified solutions onto the data preparation challenges as per the developed taxonomy,
- A set of recommendations on how to overcome the identified data preparation challenges.

The key contributions of this study are expected to help improve the state of the art and the state of the practice of data preparation for SVP models. The findings can raise awareness and understanding about the important challenges of SV data preparation; such understanding will likely assist to avoid the challenges and improve the reliability of SVP models. Furthermore, we provide recommendations to guide the future research on data preparation and data quality assessment for SVP models. Such future research outcomes are expected to ultimately result in more reliable and trustworthy SVP models. The findings from this study can also be leveraged for enhancing the existing or developing new tools for supporting the construction and application of SVP models in general, and the data preparation for SVP models in particular.

The rest of this paper is organized as follows. Section 2 describes the related work and existing SVP reviews. Section 3 presents the methodology we use to conduct our SLR. The findings of our study are presented in Sections 4, 5 and 6. In Section 7, we provide recommendations for future SV data considerations and research. Finally, in Section 8, we state the applicable threats to validity of our findings, and conclude our study in Section 9.

## 2 Background and Related Work

SVP is a data-driven process that uses learning-based methods to make predictions, and hence follows the standard ML workflow.

### 2.1 Data Preparation

Figure 1 displays the steps involved in a learning-based workflow [18]. For the purposes of SVP and this study, we consider labeling to occur before cleaning.

In our analysis of the reviewed papers, we focus on the data-oriented steps (data collection, data labeling and data cleaning) of the ML workflow. The first step of the ML workflow is the model requirements phase, which identifies the necessary requirements and applications of a model. For our study, we consider this step as *data requirements*, as it is necessary to identify the requirements of the data used to build a model, e.g., what kind of data will be used and from where it will be collected. Hence, these data requirements form a necessary preliminary component of data preparation. We collectively define the first four steps of the ML workflow as *data preparation*.

Practitioners have agreed that the majority of the time taken to construct an ML pipeline is consumed by data preparation [6]. In the 2019 Appen State of AI survey [19], it was reported that a majority of practitioners spend upwards of 25% of their time gathering, cleaning or labeling data. Despite their importance, data preparation processes have rarely been discussed or investigated exclusively [5].

### 2.2 Software Vulnerability Prediction

Software Vulnerability Prediction (SVP) models aim to automatically learn SV knowledge and patterns from historical data. This knowledge can be used to make predictions on the presence of SVs. This process was first noticeably conceptualized in 2007 by Neuhaus et al. [20], and has seen continual technical advancement through research efforts [4].

SVP can be considered as an early form of software security quality assurance, as a trained model can make predictions quickly on static code artefacts, without the need for compilation. In this sense, SVP has been compared to static application security testing methods [21]. Ghaffarian and Shahriari [22] categorized SVP methods into two main approaches: models that do not analyze program syntax and semantics, and models that do. The former utilizes software metrics to describe the code modules of interest, whereas the latter perform directly on source code tokens to perform vulnerable code pattern recognition. Due to the rising popularity of Deep Learning (DL) methods [4], researchers have focused more heavily on approaches that do analyze program syntax and semantics, through the use of text-based, sequence-based or graph-based source code feature representations [13]. Additionally, there are hybrid approaches that utilize both software metrics and code tokens [23].

Data preparation for SVP follows the standard workflow for ML data preparation. SV labels are assigned to the extracted code modules to obtain a labeled SV dataset [10]. The process is heavily dependent on the data sources selected for the codebase and SV labels. Figure 2 displays the SVP data preparation pipeline.

### 2.3 Existing SVP Reviews

With the increasing popularity of data-driven approaches for software vulnerability analysis and discovery, several researchers have reviewed the published SVP approaches and techniques. We briefly describe the key focus areas of the relevant review studies below.

Three papers by different groups of researchers, Li and Shao [24], Coulter et al. [10], and Ghaffarian and Shahriari [22], reported separate reviews of the literature on the use of machine learning and data mining for software vulnerability discovery and analysis. Coulter et al. [10] provided a more general framework for data-driven cybersecurity tasks, including SVP, whereas Li and Shao [24] and Ghaffarian and Shahriari [22] focused on the specific features and approaches. Le et al. [25] also conducted a survey of data-driven methods for SV assessment and prioritization,

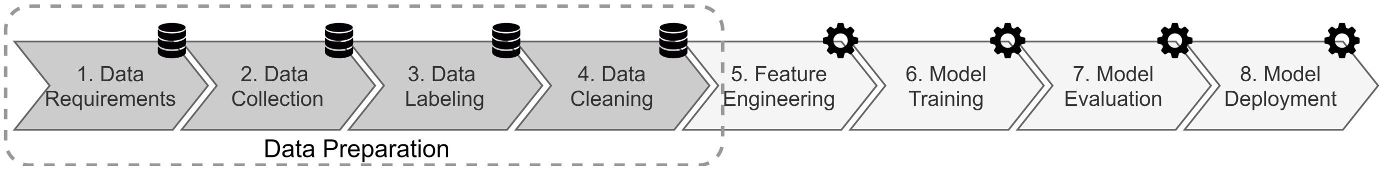

Fig. 1. The machine learning workflow. Adapted from Amershi et al. [18].

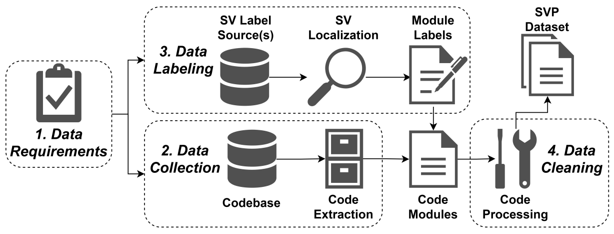

Fig. 2. The SVP data preparation pipeline.

but they did not consider SV discovery. With the success of DL in fields such as image processing, speech recognition, and natural language processing, researchers have been increasingly motivated to apply DL for the SVP domain. Lin et al. [13], Singh and Chatuvedi [26], and Zeng et al. [16] all conducted an analysis of the deep learning techniques used by researchers for SVP.

To the best of our knowledge, only two studies have been published that focus on *systematically* reviewing SVP research and knowledge: Semasaba et al. [17], and Hasif et al. [4]. The former exclusively investigated Deep Learning techniques, whereas the latter provided a wider view of SV detection, including non–learning based techniques. Similar to the previous secondary studies, the analysis of these systematic reviews focused on the models, techniques and features. Furthermore, all existing secondary studies for SVP focused on the model-oriented steps of the ML workflow, particularly features and techniques (steps 5-6 of Figure 1).

To this extent, there has been a little focus on the data-oriented processes. The SV data used to train a model is the most imperative component of this data-driven process. Although most studies have reported data preparation and data quality as significant issues for this research area [4], [10], [13], [16], [17], [24], they have not performed in-depth analysis of the data quality in SVP research to determine the encountered issues or potential solutions. This knowledge gap fails to provide practitioners and researchers with the specific insights needed to remediate data quality issues. It is vital to gain a better understanding of the quality of data utilized for SVP research; such comprehension is also expected to improve our abilities to better understand how well the SVP approaches work in practice.

Hence, an effort like ours can be of great importance as it not only highlights the critical research gap, but also contributes to the evidence-based body of knowledge of data preparation for SVP models. Whilst existing reviews have yielded important insights into this research domain, our systematic review has been motivated by several unique research questions whose answers have enabled us to provide novel findings and potentially useful insights. The knowledge produced by our SLR can be an important complementary piece to the existing secondary studies for providing a consolidated picture of the published literature on different components of the SVP pipeline.

## 3 Research Methodology

To obtain insights into the SVP data preparation processes, challenges and solutions, we conducted an SLR of SVP literature. Our findings will potentially be useful to both researchers and practitioners for providing guidance for future SV data preparation and in assessing the validity of existing SV datasets.

To conduct this SLR, we followed the methodological guidance provided by Kitchenham et al. [27] and Zhang et al. [28] to ensure that our assessment of the existing literature was unbiased and repeatable. The research method was conducted in close collaboration by the first two authors, with guidance from the third author.

Figure 3 presents the complete search and study selection workflow, and the number of retrieved papers at each stage. The search and study selection process was conducted in February 2021. We obtained a total of 61 studies from our study selection process, which are presented in the Appendix.

To guide our analysis, we aimed to address the three Research Questions (RQs) presented in Table 1.

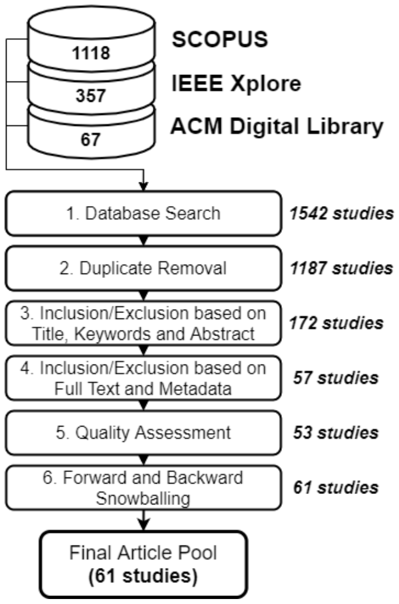

Fig. 3. Stages of the study selection process.

TABLE 1
Research questions addressed in this study.

| Research Question (RQ) | Motivation |
|---|---|
| **RQ1.** What considerations do researchers make for each of the data preparation processes when constructing SVP datasets? | We investigate the SVP data preparation practices and choices reported by researchers to help inform the state of the practice and reasoning. The embodiment of this knowledge helps provide workflow guidance for researchers and practitioners, and assists them to identify the important decisions and reasoning when selecting or building an SV dataset. |
| **RQ2.** What are the considered challenges and issues for SV data preparation and datasets? | We aim to analyse the reported data challenges to provide an overview of the issues that researchers face when performing SV data preparation or using SV datasets. The results of this RQ will help guide the decisions made by researchers when selecting their data preparation steps, and inform practitioners of the current issues and limitations of the reported empirical SV analysis and prediction. |
| **RQ3.** How do researchers address dataset issues and preparation challenges? | This RQ builds upon the findings of RQ2 to analyse the remediation techniques that researchers have used to help overcome the aforementioned challenges. Hence we not only provide a categorization of data challenges for SV-related research, but we also map the solutions that researchers have used to address these issues. These findings can help researchers and practitioners overcome data challenges in the future. |

TABLE 2
Formulation of the search string.

| Category | Subject | Search Terms |
|---|---|---|
| ***Population*** | Software | *"software" OR "code"* |
| ***Intervention*** | Machine Learning | *"learn" OR "neural network" OR "artificial intelligence" OR "AI-based" OR "predict"* |
| | Static Application | *NOT ("fuzz" OR "test" OR "attack" OR "adversarial" OR "malware" OR "description")* |
| ***Comparison*** | - | - |
| ***Outcomes*** | Software Vulnerability Prediction | *"vulnerability" AND ("predict" OR "detect" OR "classify" OR "identify" OR "discover" OR "uncover" OR "locate")* |

## 3.1 Search Strategy

We began with a search strategy to extract all potentially relevant research papers from academic digital libraries.

To design the search string, we utilized the PICO (Population, Intervention, Comparison, Outcomes) framework [29]. The *Comparison* component was not applicable to our review because our goal was not to conduct comparison of software with different interventions. Table 2 presents the key terms for each PICO component; we formed our search strings through the union (AND) of the PICO components. We altered the search string suitably to match the differences in the search capabilities of each database. When applicable, we matched the relevant keywords in the title, abstract and keywords of the papers, except for exclusion keywords (prefaced with NOT) which were only matched in the title. Wildcard matching was performed to capture different word variants when available; otherwise we defined the term variants manually, e.g., *predict* and *prediction*. When available, we applied additional search filters to match the exclusion criteria defined in Section 3.2.1 (i.e., limiting to English articles or research papers.) Table 2 only defines the base strings. To find these strings, we consulted papers included in the previous reviews. The full search strings are available in our online appendix[1].

We applied this search string to the two most frequently used academic digital libraries for software engineering, as identified by Zhang et al. [13]: IEEE Xplore, and ACM digital library. We then additionally included SCOPUS as it is the largest academic literature database available [25], which indexes several other smaller academic databases. We did not use other search engines such as Google Scholar due to the amount of noise in the search results and need for subjective stopping conditions. We initially retrieved 1542 studies: 1118 studies from SCOPUS, 357 studies from IEEE Xplore, and 67 studies from ACM Digital Library. We downloaded all retrieved studies and then manually removed duplicates, which reduced the total number of studies to 1187.

[1]https://github.com/RolandCroft/SVP_Data_SLR_Appendix/blob/main/Search_Strings.md

## 3.2 Study Selection

We sought to select any paper on the topic of Software Vulnerability Prediction (SVP), which we have defined as any model utilising supervised learning-based techniques (ML or DL) for prediction, detection or discovery of an SV in a static code module. We included any study that contributes an SVP model, process or evaluation based on our definition of SVP.

Our definition of SVP hinges on three major principles: *learning-based*, *vulnerability discovery*, and *static code artefacts*. Firstly, we have defined learning-based as the use of a supervised ML or DL algorithm that can learn from training data to make predictions on a dataset [31]. To this extent, we did not include anomaly detection, unsupervised methods, or studies that focus on pure statistical or correlation analysis. Secondly, the study must have utilized a model that aims to *discover* unknown SVs within a code artefact. This excluded methods that used code clones or similarity detection, as these methods are only able to detect a pre-defined set of SVs and are unable to discover new types. We also did not consider malicious code as SVs. Thirdly, we only included studies that used static code artefacts to make predictions; either source code, code binaries or an intermediary representation. Hence, we excluded any study that requires runtime analysis of the code (e.g., dynamic testing or attack detection).

#### 3.2.1 Inclusion/Exclusion Criteria

The inclusion/exclusion criteria we adopted, displayed in Table 3, were inspired by similar studies [30], [31].

TABLE 3
The inclusion/exclusion criteria.

| **Inclusion Criteria** |
|---|
| I1. The study relates to the field of SVP, and informs the practice of Software Engineering. |
| I2. The study presents a unique SVP process or evaluation. |
| I3. The study is a full paper longer than six pages. |
| **Exclusion Criteria** |
| E1. Solely a literature review or survey article. |
| E2. Non peer-reviewed academic literature. |
| E3. Academic articles other than conference or journal papers, such as book chapters or dissertations. |
| E4. Studies not written in English. |
| E5. Studies whose full-text is unavailable. |
| E6. Studies published to a venue unrelated to the discipline of Computer Science. |
| E7. Studies that are published to a journal or conference with a CORE ranking of less than A and H-index less than 40, *and* that have a citation count of less than 20. |

To ensure that we obtained a set of high-quality papers, we adopted a venue assessment approach (E7) used by Sabir et al. [32]. We removed the studies published in low quality venues: venue ranking below A using the CORE ranking system[2,3], and h-index below 40 as recorded in the Scimago database[4]. However, the original influential papers of this domain may have been published in low quality venues. Hence, we only excluded a paper based on venue if it had also not been cited frequently (<20 citations). The citation count is obtained through Google Scholar[5]. The first two authors collaboratively determined suitable thresholds for this criterion through an initial pilot study of 100 papers, to confirm that any papers excluded through this criterion were indeed of lower quality.

We first excluded 1015 studies using information extracted from the title, abstract and keywords. We then excluded an additional 115 studies after processing the full text and metadata (i.e., venue, article type, citations) to obtain a set of 57 studies.

#### 3.2.2 Quality Assessment

For SLRs, it is vital to assess the quality of primary studies to ensure that we form a proper and fair representation of the research works [27]. We conducted the assessment process using a quality checklist, and excluded any study that did not pass the checklist. We adopted the quality checklist defined by Hall et al. [33], and refined by Hosseini et al. [30] in their SLR of defect prediction models, as the defect prediction process shares similarities with SVP. This resulted in three stages of assessment: the data, the prediction model details, and the evaluation criteria, displayed in Table 4. Although our study only considers data preparation for analysis, we assessed all three criteria to determine the overall quality of the paper. We removed a total of four studies that did not pass the quality assessment criteria.

[2]http://portal.core.edu.au/conf-ranks/
[3]http://portal.core.edu.au/jnl-ranks/
[4]https://www.scimagojr.com/journalrank.php
[5]https://scholar.google.com.au/

TABLE 4
The quality checklist.

| **Data Criteria** |
|---|
| DC1. The data source must be reported. If a publicly available dataset is used, the name must be reported. |
| DC2. A description of the data, such as its size, programming language and class distribution, must be provided. |
| DC3. The process in which the independent variables are extracted from the data as input to the model must be clearly stated. |
| DC4. The method in which the data is labeled as vulnerable and non-vulnerable must be clearly stated. |
| **Prediction Model Details Criteria** |
| MC1. The output of the model must be clearly defined. |
| MC2. The granularity of the dependent variable(s) must be reported. |
| MC3. The machine learning method and approach must be clearly reported. |
| **Evaluation Criteria** |
| EC1. The performance measure of the model must be reported. |
| EC2. The predictive performance values must be clearly presented in terms of raw performance numbers, means or medians. |

#### 3.2.3 Snowballing

It is expected that an initial automated search strategy will be unable to identify all relevant studies, as the search string cannot identify obscurely phrased studies, and the digital libraries selected do not exhaustively include all peer-reviewed literature [34]. Hence, after we conducted initial study selection, we utilized manual search processes, both forward and backward snowballing, to obtain additional relevant studies that were not contained in our selected digital libraries or identified by our automatic search. Forward and backward snowballing identify additional relevant studies from papers that cite or are included in the reference lists of the set of included studies, respectively [34]. These identified papers were similarly assessed using the inclusion/exclusion criteria and the quality assessment criteria. We included an additional eight papers in the final set through the snowballing process.

Our final article pool contained *61 studies*; 53 studies which passed the initial selection process and eight additional snowballed papers. The studies are listed in the Appendix.

### 3.3 Overview of the Primary Studies

Figure 4 displays the number of selected SVP papers over the years. We have not reported values for 2021 as this data is incomplete. We observed that this area of research has received exponential popularity within the last two years. This indicates that this is an area undergoing huge growth at the time of this study. Hence, our review contributes important and timely value to this emerging area by synthesizing the current knowledge of the underlying data preparation processes for these empirical studies.

### 3.4 Data Analysis

#### 3.4.1 Data Extraction

We used the a data extraction form and data extraction processes outlined by Garousi and Felderer [35] and Kitchenham et al. [27]. Our data extraction form, provided in our

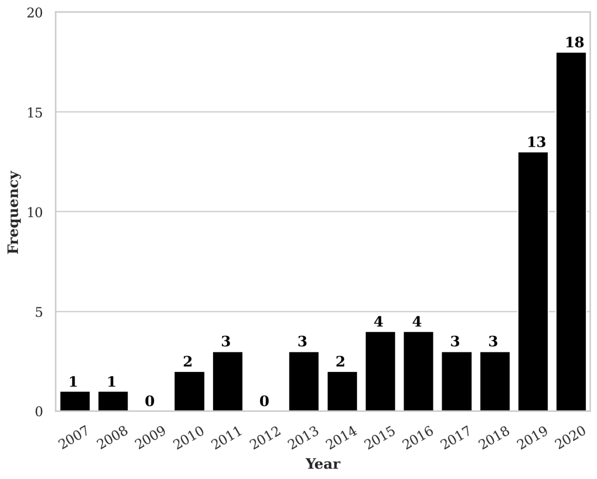

Fig. 4. The number of selected primary studies by year.

online appendix[6], consisted of 50 fields describing all data related steps reported by the authors and the details of their dataset(s). These fields consisted of five checkbox questions, 29 multiple choice questions, nine short answer questions, and seven long answer questions. Thirty seven of the 50 fields pertained to the first RQ, and the other 13 collectively related to the latter two RQs. The data extraction form was completed collaboratively using Google Sheets.

An initial pilot study of 10 papers was conducted collaboratively by the first two authors to help design the form and ensure author agreement [35]. The first two authors then performed data extraction individually; the paper set was divided in half randomly for each author to complete. After this process was completed, each author reviewed the data extraction outputs of the other author to ensure consistency. Disagreements were resolved through discussion.

#### *3.4.2 Data Synthesis*

The aim of an SLR is to aggregate information from primary studies [27]. For RQ1, we qualitatively examined the outputs of our data extraction form to identify and report the major factors relating to each of the four data preparation steps.

For RQ2 and RQ3, we used thematic analysis to synthesize the data [36]. Specifically, this process was used to identify the data challenges and solutions reported in the primary studies. Any discussion in a paper that explicitly had mentioned a challenge pertaining to the data, resolved or unresolved, was coded. To ensure that this qualitative coding was grounded by the data, and not affected by any biases of the data extraction form, we imported the full papers into Nvivo [37], a qualitative data analysis tool, and performed coding on the papers directly. We followed the steps for thematic analysis developed by Braun and Clarke [36]:

1) *Familiarizing with data*: The initial familiarization was done through the data extraction phase (Section 3.4.1) in which the first two authors read each full paper and filled the data extraction form. This familiarized the first two authors with the relevant factors relating to SV data that were discussed in the papers.
2) *Generating initial codes*: To generate initial codes, we used *open coding* of the relevant text in the primary studies using Nvivo. The data was broken down into smaller components and labeled using a code [36], where a code is a word or phrase that acts as a label for a selection of meaningful text in the paper. This process was completed iteratively, with the initial codes being revised and merged in later rounds. Each primary study was usually allocated to more than one code or theme, as each paper can discuss multiple SV data challenges and coding was done on small individual components of the papers.
3) *Searching for themes*: We reviewed all the codes and sorted them into themes. As data challenges revolve around data quality, we used existing data quality dimensions [38], [39] to identify potential groupings that the codes might fall under.
4) *Reviewing themes, defining and naming themes*: This process involved reviewing all the codes and themes, and revising their allocations.
5) *Producing the report*: We present the findings of our thematic analysis in Sections 5 and 6.

## 4 SV Data Preparation Considerations (RQ1)

We first provide an overview of the considerations that researchers have made when performing SV data preparation processes, which we have identified qualitatively through our data extraction process. This documentation of the considerations helps to inform practitioners and researchers of the state of the practice. Furthermore, it can assist these users to better understand how to construct an SVP dataset, and the important aspects to scrutinize. Figure 5 displays the main decisions that need to be made for each data preparation step.

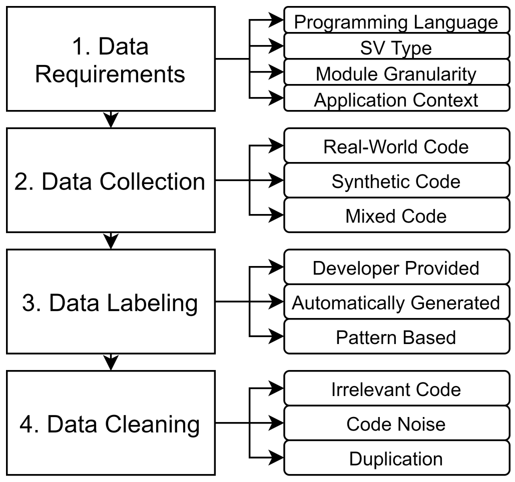

Fig. 5. SVP data preparation step considerations.

### 4.1 Data Requirements

In the data requirements phase, the requirements for the data to achieve the desired model context and capabilities

[6]https://github.com/RolandCroft/SVP_Data_SLR_Appendix/blob/main/Data_Extraction_Form.pdf

are specified. There are four main components of the data requirements that need to be specified for SVP:

**Programming Language(s).** Researchers are motivated to explore different approaches to mitigate security risks aroused for different languages. As seen in Figure 6, C/C++, PHP and Java have been the most commonly investigated languages among the primary studies.

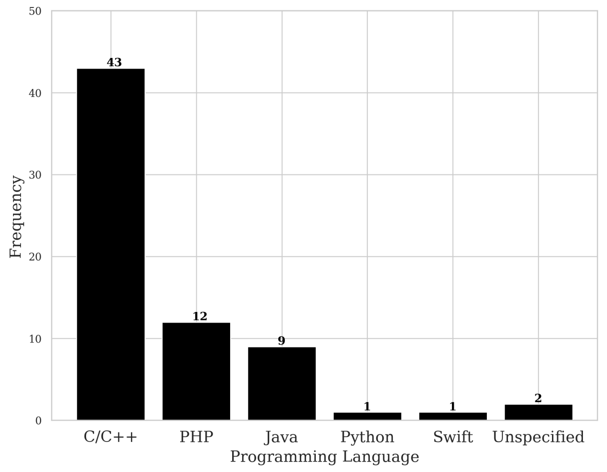

Fig. 6. The number of primary studies for each programming language.

C/C++ has been most frequently chosen for analysis as it has a lower level of abstraction and is commonly used to build security critical applications [P4, P16, P54, P58]. PHP has been commonly used for programming web applications, which are highly susceptible to vulnerabilities and exploits [P1, P6, P25, P41, P50, P55], and hence researchers have aimed ensure its security. Java has also been commonly chosen as it is overall one of the most popular programming languages [P39, P40, P46].

**Vulnerability type(s).** Similar to the previous consideration, researchers may target detection capabilities towards certain SV types. However, we observed that the majority of methods are capable of detecting a variety of SVs. Over 45% of studies (28 out of 61) did not even report the types of SVs present in the data, and researchers were often limited to the SV types present in their SV label source. However, some studies chose to restrict their analysis to more critical SVs of interest. For example, Fidalgo et al. [P6] and Shar and Tan [P25] focused their analysis on SQL injection and cross-site scripting (XSS) as these are common critical web application vulnerability types. Wang et al. [P29] and Ghaffarian and Shahriari [P39] limited their studies to just the CWE Top-25 vulnerabilities[7]. Saccente et al. [P34] identified that a model trained to predict any SV type produces unreliable predictions in comparison to a model trained to predict just one type. However, the impacts of this data requirement decision on model efficacy are otherwise largely under-explored.

**Code module granularity.** The granularity of vulnerability detection has significant impact on a model and data collection. Depending on the granularity of the inputs used for an SVP model, it can either be used to direct testing efforts by predicting which large scale components are potentially at risk [15], or to explicitly detect fine-grained components that contain vulnerabilities [24]. In our set of primary studies, we identified six levels of granularity (in descending order of granularity): 1) component level, 2) file/class level, 3) function/method level, 4) program slice level, 5) statement level, 6) commit level. In Figure 7, other than the component-level, the larger granularities are shown to be of more popularity. The file level has been considered as the standard granularity for SVP research [P12, P36, P46] and has been confirmed as actionable by developers [P5]. However, researchers have recently begun to favor finer granularities as they better enclose the scope of vulnerable code snippets, and are more easily inspected [P2, P3, P10, P15, P31, P32].

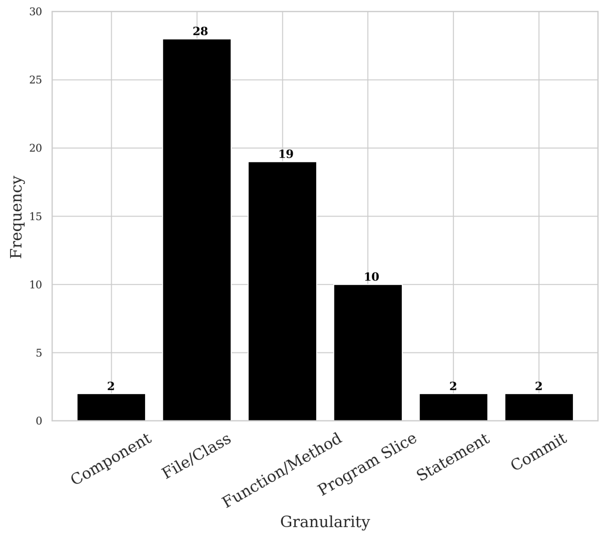

Fig. 7. The number of primary studies for each level of granularity.

**Application contexts.** There are three main vulnerability detection application contexts: within-project, cross-project, and mixed-project, in which each have different requirements. For within-project prediction, both the training data and the testing data come from the same project. In contrast, for cross-project prediction, there is an assumption that there is an insufficient amount of training data available in the target project. Therefore, labeled data from other source projects are used for training. More than one dataset can be collected to compensate for the inadequacy of labeled data in the target project. Mixed-project prediction is a special case of the cross-project setting. The labeled data from multiple projects are combined together to produce sufficient data for both model training and testing. This differs from the cross-project setting in which data from other projects only comprise the training dataset.

Within-project prediction has proven to be the more popular prediction context, with 52% (32 out of 61) of the primary studies forming their datasets from a singular project. This is because researchers have considered within-project prediction to be the standard use case of SVP [P5, P36]. Furthermore, cross-project prediction has often performed poorly due to differences in data distribution of the source and target project(s) [P1, P12, P26, P39, P56]. Only 18 of the primary studies considered cross-project prediction, 13 of

[7]https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

which also considered within-project prediction. However, due to the various data issues that we will discuss in Section 5, several researchers have considered the reuse of existing datasets from other projects to be a necessity [P1, P10, P18, P21, P22, P26, P31, P37, P41, P43, P56]. Twenty three of the primary studies (38%) performed mixed-project prediction and did not exhibit consideration for the projects that form their dataset.

### 4.2 Data Collection

SV datasets can be categorized into three main areas based on the type of data sources used to generate the code modules: real-world data, synthetic data, and mixed data. The type of data is the main influence on how the data is collected.

***Real-world data.*** Both the code and the corresponding vulnerability annotations have been derived from real-world repositories. The code has been typically collected for projects hosted on repository hosting sites, such as GitHub [40], or through a different version control system. Experiments conducted using this data are usually considered to be better representative of industrial application because of the reflection of the complexities of the real-world vulnerabilities [41].

***Synthetic data.*** The vulnerable code examples and the labels have been artificially created. The examples from these data sources are synthesized using known vulnerable patterns. Synthetic datasets include SARD [42], OWASP Benchmark [43], and SQLI-Labs [44]. These datasets were originally used for evaluating traditional static and dynamic analysis based vulnerability prediction tools, due to their large test suite size and noise-free information.

***Mixed data.*** Several researchers have opted to create datasets by merging both real-world and synthetic data sources [P32, P44, P48]. This is typically done to achieve a sufficient dataset size whilst maintaining a certain level of real-world representation. Mixed datasets have been primarily constructed for DL-based studies [P32, P44], which are particularly data hungry in comparison to ML.

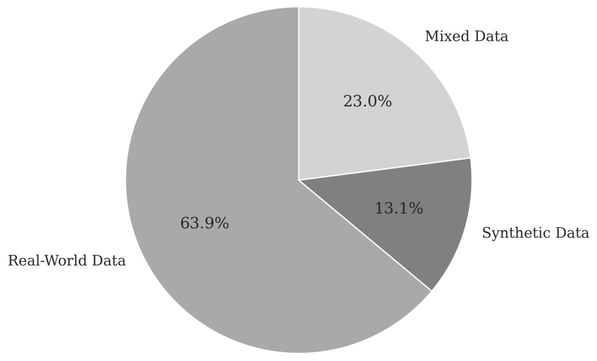

Fig. 8. The proportion of primary studies for each data source type.

Figure 8 displays the number of primary studies that use each data type. Real-world datasets have been considered the *de facto* type of data sources in this domain by researchers, primarily as they better represent real-world scenarios than synthetic examples [41]. The representativeness of a dataset is an important consideration as it helps improve the generalizability and validity of findings [45]. Studies have further claimed that the code patterns of synthetic test cases follow a similar coding format, failing to reflect the characteristics of code patterns in production environments [P2 ,P3, P9, P29, P31, P39, P42, P43, P61].

However, there are two primary positive traits of synthetic datasets. Firstly, there are a much larger number of labeled synthetic samples that are able to be created in comparison to real-world examples [P8, P23, P34, P38, P39, P44, P54]. SVP is a data-hungry process that requires a sufficient amount of training data [4]. The existence of their labels also significantly reduces the effort of data preparation [P3, P6, P10, P39, P40]. Secondly, as the code samples are generated with their labels, the labels are more clean and reliable than data extracted from noisy real-world repositories [P39, P40], for reasons discussed in the following section.

### 4.3 Data Labeling

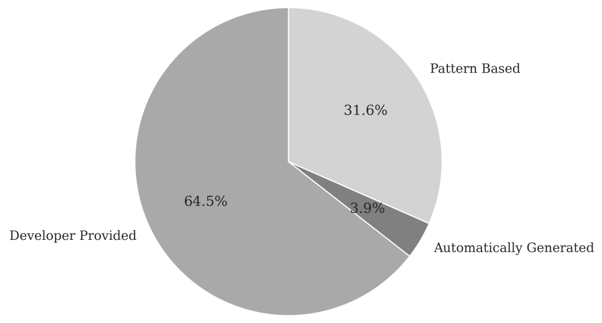

Fig. 9. The ratio of SV data label sources. A study may use more than one label source.

For SV data, labels categorize whether a code module is vulnerable or not. Data labeling involves extracting data labels using an external source or tool, to assign to the collected code modules. We observed three SV label sources, that align with the findings of Chakraborty et al. [41]: developer-provided, automatically generated, and pattern-based. Figure 9 displays the ratio of these three SV label types. The choice of labeling approach is often dependent on the data source type collected, as outlined in Section 4.2. Hence, the relative frequency of each method is similar to that of Figure 8.

Figure 10 displays the labeling process using developer provided labels. These labels have been extracted from vulnerabilities that have been identified by developers and reported in security advisories or issue tracking systems, such as NVD [46], Jira [47] or Bugzilla [48]. Whilst these information sources are usually accurate, it is not the same as developers hand-labeling code modules, as patches do not directly equate to labels. Researchers have expended significant effort to trace the label source to the code modules [P2, P18, P29, P43, P61]. This involves identifying the relevant vulnerability reports, extracting code fixes, and

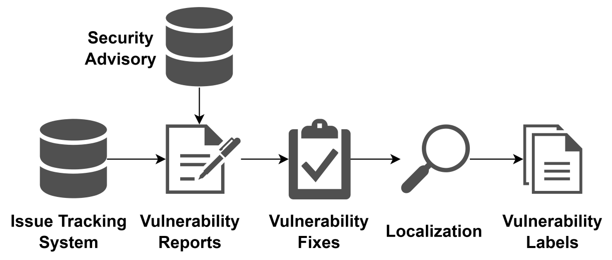

Fig. 10. The SV labeling process using developer provided SV labels.

localizing these fixes to the relevant granularity and code modules, with each step introducing potential noise to the labels. Labeling of the non-vulnerable class is also quite subjective in this scenario, as there is no associated label source for this class. Consequently, it is a form of weak supervision [49] that can introduce inadequacies into the data.

Automatically generated labels have not relied on a third-party label source, but instead have used an additional oracle to provide labels directly for the collected code modules. Two of the primary studies used static analysis tools to provide labels to code datasets [P37, P46]. This approach is noticeably the least considered as it heavily relies on the accuracy of the oracle used for labeling.

Pattern-based labels have been obtained for synthetic data sources, that use SV patterns to generate both the code modules and labels. The goal of an SVP model applied to this synthetic data and label source, is to re-learn the patterns that generated the code modules of each class. These labels are largely considered noise-free, unlike the other two approaches, as the modules and labels are inherently connected.

### 4.4 Data Cleaning

Data cleaning is the fourth step of data preparation. Whilst data cleaning is important, it is the only data preparation step that is non-essential. Collected code can be labeled and used immediately if it is extracted in an appropriate format. Hence, we observed that not all of the primary studies have discussed this step; nor has it been discussed in as much detail as the other steps.

Code modules are the raw data for SVP, so the data cleaning step involves processing collected code modules so that they are in an appropriate format for feature extraction. Table 5 lists the common data cleaning approaches that we observed from the reviewed primary studies. The cleaning approaches fall under three issue types: irrelevant code, code noise, and duplication.

**Irrelevant Code.** Firstly, irrelevant code has been commonly removed from the collected code modules. For all of the primary studies, this involves removing any code that was not of the target programming language, and removing any code that did not fall under the target granularity, such as code not contained in functions for the function-level granularity. Some studies also removed code that was not of relevance to the project or not at risk, such as test cases, third-party code modules, code scripts, and make files [P1, P4, P30, P56].

TABLE 5
Common data cleaning approaches for SVP datasets.

| *Issue* | *Cleaning Approaches* |
|---|---|
| **Irrelevant Code** | • Remove code that is not of the target programming language(s).<br>• Remove code that is not of the target granularity.<br>• Remove irrelevant code files: test cases, third-party-code, scripts and make files. |
| **Code Noise** | • Remove blank lines, non-ASCII characters and comments from the code.<br>• Ignore code with syntax issues or errors.<br>• Replace the user-defined variable and function names with generic tokens. |
| **Duplication** | • Remove highly correlated items.<br>• Remove the duplicated code. |

**Code Noise.** Secondly, some studies have further processed the collected code to remove the potential noise in the data that may impact the created features. These steps are only of value to the studies that extract the features directly from the code tokens, while software-metrics are usually robust to such code noise. Several studies removed comments, blank lines and non-ASCII characters [P1, P10, P20, P28, P32, P34, P44, P56], as these are not relevant to SVs. Some studies also replaced user specific tokens, such as user-defined variables, function names, and string literals, with generic tokens to increase the homogeneity of the features [P7, P23, P28, P29, P32, P50, P56]. Some studies also removed the code that they found to have syntax issues or errors [P9, P50], as this may later impact the feature extraction efforts.

**Duplication.** Thirdly, several studies have attempted to remove the duplicated code modules, as duplicate entries may introduce bias into a produced model [50]. Duplicate code modules can be present due to multiple of the same code file, snippet, or software version being collected [P1, P8, P15].

## 5 Data Challenges (RQ2)

This section identifies the data challenges that researchers have reported in the primary studies, in relation to the data preparation steps. As discussed in Section 3.4.2, we used thematic analysis to analyze the data quality issues that researchers have explicitly reported. These issues were coded, revised and merged by the first two authors of this study. The themes that we have identified are inspired by existing data quality dimensions [38], [39], as data challenges revolve around data quality. Table 6 details the key data challenges and considerations that we observed through our analysis. The data challenges can be summarized as pertaining to data Generalizability, Accessibility, labeling Effort, Scarcity, and both Label and Data Noise.

### 5.1 Generalizability

Generalizability describes the ability for data to extend to other contexts, both in terms of findings and application [39]. Hence, this largely measures the external validity of the produced analysis, based on the data. This challenge primarily involves the data requirements step, as this is

TABLE 6
Taxonomy of SV data challenges identified from the primary studies.

| Theme | Challenge | Key Points | Paper ID | # |
|---|---|---|---|---|
| *Generalizability* | **Ch1:** Real-World Representation | • Synthetic data is not representative of real-world code | P[2-3, 9, 29, 34, 39, 60] | 43 |
| | **Ch2:** External Generalization | • Data may be language specific | P[1-2, 11, 15, 20-23, 26, 30, 32, 38, 41-42, 44, 46-47, 55-57, 60-61] | |
| | | • Data may be application or domain specific | P[1, 4-5, 11-13, 19-22, 30, 35-36, 40-42, 46, 52, 55-57, 59] | |
| | | • Data may be specific to vulnerability type | P[15, 24, 26, 32, 38, 61] | |
| | **Ch3:** Completeness | • SVs may span code modules | P[1, 3, 7, 10, 15, 21, 31, 45] | |
| | | • Code representation may have limited scope | P[1, 3, 8, 13, 32, 44] | |
| | | • SVs may be present in non-targeted code files | P[1, 12, 20, 46, 53, 57] | |
| *Accessibility* | **Ch4:** Cold-Start Problem | • SVs are required to have originally occurred | P[4, 10, 21-22, 31, 37, 41, 56] | 25 |
| | **Ch5:** Data Entry Availability | • Not all data entries are obtainable | P[1, 5, 11-13, 20-21, 30, 45, 49, 56] | |
| | | • Usage of Version Control Systems is unstable | P[1, 12, 56] | |
| | **Ch6:** Data Privacy | • Source code and SV data are required to be available | P[9-10, 16, 26-27, 32, 37-38, 45, 48, 61] | |
| | | • Security advisories can be private or vague | P[1, 13, 56] | |
| *Effort* | **Ch7:** Labor Intensive | • Manual labeling is highly time-consuming | P[2, 10, 18, 26, 29, 31, 39, 42-43, 61] | 14 |
| | **Ch8:** Expertise Requirements | • Manual labeling requires high expertise | P[2, 18, 31, 43] | |
| | | • Vulnerabilities are difficult to identify | P[20, 30, 33-34, 42-43] | |
| *Data Scarcity* | **Ch9:** Data Imbalance | • Vulnerable samples are the extreme minority | P[1, 4-5, 8-9, 11-12, 14-15, 17, 19-20, 22, 24, 30-31, 33, 36, 40, 45, 50, 53-56, 60] | 32 |
| | **Ch10:** Number of Samples | • Low number of vulnerability samples | P[1, 4, 10-11, 15, 20, 23, 29-31, 33-34, 36, 45-47, 50] | |
| *Label Noise* | **Ch11:** Incomplete Reporting | • Latent, dormant or unresolved SVs can exist in the dataset | P[1, 4, 11-13, 16, 19, 21, 28, 30, 36-37, 47, 56, 59-61] | 31 |
| | | • SVs can be silently patched | P[4-5, 12, 30, 33, 36, 40, 47, 52, 56] | |
| | **Ch12:** Localization Issues | • Commit noise causes localization issues | P[1, 16, 29, 42] | |
| | | • Data noise causes localization issues | P[2, 8, 13, 28, 32, 44] | |
| | | • Bug reports do not document code location | P[28, 30, 33, 42, 56] | |
| | | • Version tracking is complex and erroneous | P[1, 24, 30, 56] | |
| | **Ch13:** Erroneous labeling | • Manual labeling can be inaccurate or subjective | P[11, 18, 40, 43, 52, 56] | |
| | | • Static analysis tools label modules inaccurately | P[2, 46] | |
| | | • SVs may not actually be exploitable | P[1, 52] | |
| | | • Label quality is unknown | P[24, 28, 32] | |
| *Data Noise* | **Ch14:** Code Noise | • Source code has stylistic differences or syntax issues | P[7, 9-10, 13, 18, 20, 23, 28-29, 32, 34, 38, 41, 50-51, 61] | 35 |
| | | • Binary code is noisy | P[27, 48, 53] | |
| | **Ch15:** Redundancy | • Some entries are indistinguishable between classes | P[49] | |
| | | • Code versions and localization can add redundancy | P[1, 8, 12, 15, 19-20, 24, 30, 42, 46] | |
| | | • Vulnerable samples have limited diversity | P[14, 28, 40] | |
| | **Ch16:** Heterogeneity | • Data contains outliers | P[25, 42] | |
| | | • Poor cross-project performance | P[1, 12, 26, 39, 56] | |

the phase where researchers determine the nature of their dataset.

***Ch1:*** *Real-World Representation.* Most of the challenges described in Table 6 arise when using real-world data. Consequently, several researchers have opted to use synthetically created data to construct their models, as described in Section 4.2. Synthetic datasets are artificially crafted to address the data challenges present in the real-world data; these data sources are accessible, large, low-effort to use, and the labels have less noise. As such, they are an attractive option for researchers.

Given synthetic data may not represent the real-world data, it is considered a big limitation that may render such a dataset unusable unless this limitation is addressed. Synthetic vulnerability examples are considered to be simpler, isolated, less diverse and cleaner than real-world vulnerabilities [P2, P3, P9, P29, P34, P39, P60]. Zheng et al. [P29] found that the use of synthetic data sources may significantly inflate the reported model performance in comparison to the models using real-world code. Hence, a model trained using synthetic data is unlikely to be able to detect complex real-world vulnerabilities, which require much deeper semantic understanding and reasoning [41]. Thus, real-world data is more commonly used data source, as seen in Figure 8.

***Ch2:*** *External Generalization.* Nearly all studies face external threats to validity of their findings inferred from a specific dataset. In terms of SVP research, this relates to the limited application scope of the selected study datasets. Datasets may be specific to, and hence have troubles generalising outside of: programming language [P1, P11, P15, P20, P21, P22, P23, P25, P26, P30, P32, P38, P41, P42, P44, P46, P47, P55, P56, P57, P60, P61], application or domain type [P1, P4, P5, P11, P12, P13, P19, P20, P21, P22, P30, P35, P36, P40, P41, P42, P46, P52, P55, P56, P57, P59], and SV type [P15, P23, P26, P32, P38, P61].

***Ch3:*** *Completeness.* Completeness is achieved when a dataset has all the relevant parts of an entity's information, which is sufficient to represent every meaningful state of a real-world system [39]. However, the selected data for analysis can have a limited scope of the overall system, which makes their application context limited. This data-oriented consideration serves as a challenge for SVP, as it prevents these models from forming a "complete" solution. Firstly, if the selected granularity of code modules is too fine, researchers are forced to ignore vulnerabilities which span multiple modules [P1, P3, P7, P10, P31, P45]. For instance, function-level prediction is unable to predict more complex SVs that span multiple functions. The selected semantic representation of data may also not consider all sources of weaknesses in a software system [P3, P8, P32, P44]. For instance, Tian et al. [P3] and Li et al. [P8] only consider code snippets of library and API function calls, which would not cover all potential SVs in a system. Static source code is also unable to capture certain necessary dynamic code traits [P1, P13], such as crashes and memory leaks. Similarly, vulnerabilities may be present in code modules which are not of the target programming language of analysis [P1, P12, P20, P46, P53, P57]. In the modern development landscape projects commonly utilize multiple programming languages [51], but SVP models are largely targeted towards a single programming language of choice [P53].

## 5.2 Accessibility

Accessibility describes the ability to retrieve or obtain data from the target data sources [52]. Challenges arise from difficulties in accessing the data, either of the raw code modules during the data collection step or of the data labels during the data labeling step.

***Ch4:*** *Cold-Start Problem.* The cold-start problem is an issue originating from recommender systems in which a system is unable to draw inferences about incoming modules for which it has not yet gathered sufficient information [53]. In terms of SVP, the cold-start problem has been particularly present, as to make future predictions, we require vulnerabilities to have originally occurred and to have been documented [P4, P10, P31]. This makes SVP largely infeasible for new or immature organisations [P10, P21, P22, P31, P37, P41, P56]. Furthermore, the acquisition of initial high-quality training data is a major issue, as seen in the other challenges.

***Ch5:*** *Data Entry Availability.* Since a majority of SV data have been obtained through mining open source repositories, not every part of a system would be necessarily accessible to researchers. For instance, some source code might have been unavailable to the public for mining [P1, P11, P12, P20], or unobtainable due to other technical reasons [P5, P21, P45]. Similarly, some vulnerability reports might not have been able to be localized to code modules due to issues in the automatic or manual localization methods [P1, P13, P30, P49, P56].

Some researchers have even pointed out that the reliance on a version control system to track code modules causes issues in itself, as consistent usage of a version control system is unstable [P1, P12, P56]; version control systems were only widely adopted in 2005 with the introduction of *git*, hence data before this date would be irretrievable [P12, P56]. Furthermore, organisations might switch the version control system they were using, losing previous software history [P1].

***Ch6:*** *Data Privacy.* The potential commercial sensitivity of both software code and SV reports means that organizations are often not willing to share private-source code or data to researchers [54]. This data privacy creates many data accessibility issues. Firstly, several researchers have observed that commercial systems have not provided their source code [P9, P10, P26, P32, P45, P48, P61], making SVP via source code on these systems infeasible. Furthermore, organisations and practitioners might desire to limit the availability of their security advisories by making them private or vague [P1, P13, P56]. By concealing information about vulnerabilities, it is theoretically harder for an attacker to exploit a system. Similarly, an organisation might not even maintain a security advisory or document SVs [P16, P27, P37, P38].

To represent real-world data, researchers have often surreptitiously avoided this issue through the use of open-source repositories that have public vulnerability records. Without open sources, data retrieval and reporting can become very difficult due to commercial sensitivity. Whilst this is valid, open-source data is not representative of software engineering practices as a whole; it is unknown whether the derived observations will generalize to private-source code and practices. Only two out of the 61 primary studies used private source data [P5, P11], both of which suffered from data entry availability (**Ch5**) as a result.

## 5.3 Effort

Effort describes the amount of human-effort required to label a dataset [39]. The standard approach for traditional supervised learning has been to have a subject matter expert hand-label a dataset. However, this is largely infeasible for SV data due to the extreme effort requirements. As such, many researchers have skirted around this challenge by using synthetic labeled data sources or reusing existing datasets. Consequently, this theme was actually the least mentioned by primary studies, as researchers that reused datasets often did not report or discuss the effort.

***Ch7:*** *Labor Intensive.* Labeling code or bug reports as SV-related is a non-trivial task, which makes it highly labor intensive when coupled with the sheer number of modules to examine [P2, P18, P29, P43, P61]. Dowd et al. [55] estimated that one hour of security review can cover an average of 500 lines of code. However, most modern software systems contain millions of lines of code, which makes the required man-hours infeasible. Zhou et al. [P2] stated that it took

600 man-hours to manually curate their SV dataset. Manual labeling is ultimately the most reliable labeling approach however, due to the large amount of noise for automated labeling (**Ch13**).

***Ch8:*** *Expertise Requirements.* Secure code review requires significant security expertise [P2, P18, P31, P43]. To successfully perform secure code review, a practitioner/researcher must have the capability to memorize and recognize thousands of security-related patterns and concepts [56], and this list of required knowledge is continually growing. Furthermore, it has been highly difficult to identify SVs in comparison to regular defects [P20 , P30, P33, P34, P42, P43], as they do not necessarily represent functional bugs and are thus hard to verify.

## 5.4 Data Scarcity

Data scarcity refers to the extent to which the quantity or volume of the available data is appropriate for the task at hand [38], [57]. As SVP is a data-driven process, it requires large volumes of data [50]. For SV data, this challenge largely represents the low number of real-world vulnerable examples available. Whilst codebases are often sufficiently large, the number of identifiable vulnerable modules in a codebase is relatively small [8].

***Ch9:*** *Data Imbalance.* The severe imbalance of vulnerable modules to non-vulnerable modules has been a challenge reported by many researchers using real-world datasets. To quantify this issue, we report the percentages of vulnerable modules in each dataset utilized in the primary studies, displayed in Figure 11. We display the real-world and synthetic datasets separately, as the synthetic datasets have been artificially constructed to over-represent the vulnerable class. Real-world datasets which were artificially altered to be balanced, or merged datasets consisting of both synthetic and real-world examples are excluded from Figure 11.

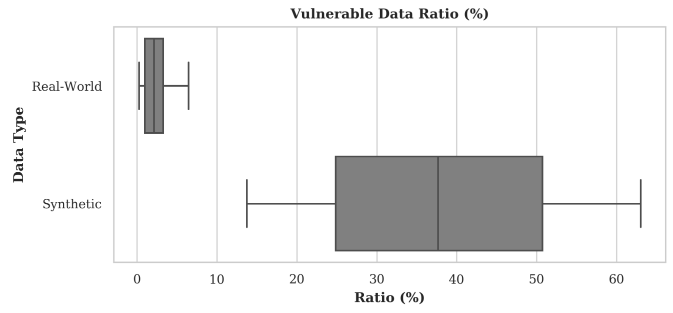

Fig. 11. The percentages of vulnerable files in datatasets utilized in the primary studies.

This has been a considerable issue for SVP research, as learning-based models are optimized to perform on balanced classes [58]. The severe class imbalance issue present in real-world SV data can lead to biased classifiers that favor the non-vulnerable class [P15, P19, P24, P30, P31, P33, P50, P54]. This challenge has been referred to as *finding a needle in a haystack* [8] and is notably unique to security defects. Shin and Williams observed that the number of reported faults was seven times larger than that of vulnerabilities [P33]. Nguyen and Tran [P24] observed that class imbalance increased as the module granularity became more fine-grained.

***Ch10:*** *Number of Samples.* Similar to **Ch9**, the severe imbalance of data leads to a very low overall number of samples for the vulnerable class. This has been a major blockade for SVP research as learning-based methods have strict data requirements; they need a large quantity of historical cases to learn from. It is common knowledge in the ML community that *more data beats a cleverer algorithm* [50]. Several studies have particularly exacerbated this issue for DL-based methods, which require even greater data size [P15, P23, P29, P34, P45, P47, P50]. This is an emerging challenge given the rise of DL-based methods for SVP [13].

## 5.5 Label Noise

Label noise relates to whether the labels in a dataset accurately represent the ground truth and are free of error [38], [54], [57]. As SV datasets have been rarely hand-labeled by subject matter experts (excluding synthetic datasets), but instead mined from historical artefacts, a large amount of noise has been introduced into the SV labels. This severely impacts the reliability of SV data.

***Ch11:*** *Incomplete Reporting.* A major problem of using historically reported vulnerabilities to label real world data is that we have only obtained a label source for the vulnerable class, and simply treated the remaining labels as non-vulnerable, creating uncertainty in the labels for the non-vulnerable class. Modules in the non-vulnerable class are not actually confirmed to be clean, just that no vulnerabilities have been historically reported. However, in reality, there can be dormant or latent vulnerabilities existing in these modules [P1, P11, P12, P13, P16, P19, p21, P28, P30, P36, P37, P56, P59, P60, P61]. Hence, the non-vulnerable class is better considered as unlabeled [P21, P47].

Similarly, researchers are limited to the use of *fixed* vulnerabilities as a label source [P52]. Unresolved vulnerabilities have been rarely disclosed as they can be exploited by attackers. Hence, during data labeling, these unresolved SVs would actually be contained in the non-vulnerable class.

Furthermore, the reliance on vulnerabilities to be reported has created a temporal issue for data collection. SVs usually take time to be detected and fixed, and hence a different number of SVs will be documented depending on the time of data collection [P21, P36]. Jimenez et al. [P36] observed that the performance of models significantly decreased when only using vulnerabilities observed before the time of model training.

Using bug reports for labeling also has created a reliance on developers or organisations to have thorough reporting practices. However, in reality, some organisations have patched some vulnerabilities "silently" [P36, P40, P56], without any documentation provided for bug reports or security advisories. Furthermore, the difference between SVs and non-security related faults can sometimes be minimal [P5, P12, P30, P33]. Hence, many security defects have not been reported by developers as such [P4, P52]. An organisation may also use multiple bug reporting systems; a reliance on just one data label source (i.e., NVD) would be incomplete [P36, P40].

***Ch12:*** *Localization Issues.* Bug reports and SV records have not always documented the location of SVs, posing a large challenge for the retrieval of SV data [P56]. Hence, researchers have often relied on the use of patches to identify the location of a vulnerability. This is flawed as not every documented vulnerability has an associated patch [P28, P30, P33, P42, P56], as it may be concealed for privacy or not yet resolved. Furthermore, patches may not always properly disclose the true location of an SV [P1], as patches may instead provide workarounds for separate modules, rather than a fix of the underlying problem. Jimenez et al. [P12] found that only 75% of vulnerability reports had an associated fix. Vulnerability reports are often incomplete and missing references [9].

Furthermore, tracking SV location from a bug fix has been non-trivial. Version information in bug reports is often unreliable [P1, P24, P30, P56], and commits can be noisy [P1, P16, P29, P42]. Identifying relevant software versions for a vulnerability is highly difficult. Due to the evolving nature of code, the assumption that all previous versions of code were also vulnerable is invalid [P24], and reporting vulnerabilities in prior versions is inaccurate as developers have little benefit from doing so [P1]. A vulnerability fix can also be buried as part of a larger commit including non-security changes. Herzig et al. [59] showed that tangled commits had significant impacts on defect labeling, with an average of 16% of files being mislabeled as a result.

***Ch13:*** *Erroneous labeling.* Label inaccuracies can additionally come from various other sources. Firstly, with noisy labels stemming from **Ch11**, as well as errors arising from localizing labels to code modules (**Ch12**), manual labeling has been a common approach to help ensure data quality. However, this approach is erroneous in itself. As discussed in **Ch8**, manual labeling is difficult and requires high expertise. Hence, it is inevitably an error-prone task [P18, P40, P43, P52]. Furthermore, the process of labeling modules as vulnerable or not can even be quite subjective [P11, P56]. There is no clear definition of the difference between a vulnerability and a fault, and hence this distinction can be nebulous to a human [P30]. For instance, if a regular function calls a vulnerable function, it is unclear whether this function should also be considered vulnerable [P1].

Some studies have utilized static analysis methods to achieve automatic data labeling without the need for historical vulnerability reports [P12, P37, P46]. However, researchers have observed these methods to be highly inaccurate and hence introducing considerable noise into data labels [P2, P46]. This process can also be flawed from a motivational perspective, as the SVP model is simply relearning the patterns used by the static analysis tool to infer the labels.

Patched vulnerabilities may not actually be exploitable in the real-world either [P1, P52]. Developers may incorrectly hypothesize security weaknesses, or err on the side of caution. This adds unreliability to the accuracy of the labels in the data source, as the SV labels may actually be benign.

Another large and open challenge has been the lack of measures to quantify label quality [P24, P28, P32]. There is no trivial way to measure or quantify the aforementioned label noise issues. Furthermore, with the reliance on historical artefacts for labeling, some sources of label noise are undetectable or unverifiable, e.g., SVs remain dormant (**Ch11**) until they have been detected. This severely impacts the reliability of SVP research, as it is unclear whether the findings have been made using valid data.

## 5.6 Data Noise

Finally, data noise refers to the noise within the raw data entries [39]; code modules for the purposes of SVP. Noise and inaccuracies in these modules may negatively affect the data and any produced features used to train a model, lowering the potential efficacy of that model.

***Ch14:*** *Code Noise.* SVP uses code as the raw data source. However, source code is noisy, which consequently impacts the effectiveness of any produced model. Developers have different coding styles and naming conventions [P28, P34, P38]. Li et al. [P61] further identified, that different projects may have different code quality due to differences in coding practices and guidelines. Real-world code can also contain syntax issues [P9, P50]. These sources of noise can severely impact the versatility of produced SVP models, as they may instead learn specificities of particular coding styles and syntax.

Furthermore, binary code is usually much noisier than regular source code. Code snippets can be difficult to trace and identify [P27, P53], or interspersed code and variables can become indistinguishable [P27, P48].

***Ch15:*** *Redundancy.* Redundancy refers to undesirable duplication in a dataset. Too much redundancy in the training data, can lead to bias and overfitting for an SVP model. The major source of redundancy in SV datasets has come from code modules having several different versions and revisions. Datasets that consider versions separately can introduce redundancy into the code entries, as the majority of the code usually stays constant between revisions [P20]. Vulnerable labels can also be duplicated over several versions, as modules can remain vulnerable for an extended period of time [P12, P19, P24, P30, P46]. Code branches are another potential source of redundancy to labels and modules as the majority of code and data is often duplicated across branches [P1, P42]. Automatic extraction of code snippets and program slices can additionally introduce duplication [P8], as duplicate program slices can be created from different entry points.

Vulnerable samples can also not be very distinct from each other [P14, P40], which limits the learning capacity of an SVP model. This is particularly present for synthetically created samples [P28]. Another issue is that vulnerable and non-vulnerable entries have limited diversity across classes [P49]. Vulnerability patches can only alter a few lines of code, making the majority of the module consistent between its vulnerable and non-vulnerable versions. This is a particularly significant issue as if the model cannot learn these subtle distinctions, it will produce high false positive/negative rates.

***Ch16:*** *Heterogeneity.* Code modules have been observed to be highly heterogeneous, which negatively impacts the diverse application of the produced SVP modules. For example, the coding conventions of one code module often do not match another, due to differences in authorship, functionality or coding style. Learning-based methods operate best when the data, especially the training and test distributions, are homogeneous so that the learnt patterns can be applied uniformly [50]. However, researchers have observed data distributions to be irregular or containing outliers, hence requiring normalization to reduce irregularities [P25, P42]. Similarly, several researchers have observed that SVP models perform poorly in a cross-project setting [P1, P12, P26, P39, P56], due to the heterogeneity of these datasets; the coding conventions and functionality of one project rarely mirror another.

## 6 Data Challenge Solutions (RQ3)

In this section, we present the various solutions that researchers have presented in the reviewed studies to solve data challenges or help improve data quality. Figure 12 displays the main areas of the solutions that we have identified. We again used thematic analysis, described in Section 3.4.2, to identify the categories of the identified solutions. We mapped the solutions to the data challenges based on the data challenge themes that the solutions were connected to when reported in the primary studies.

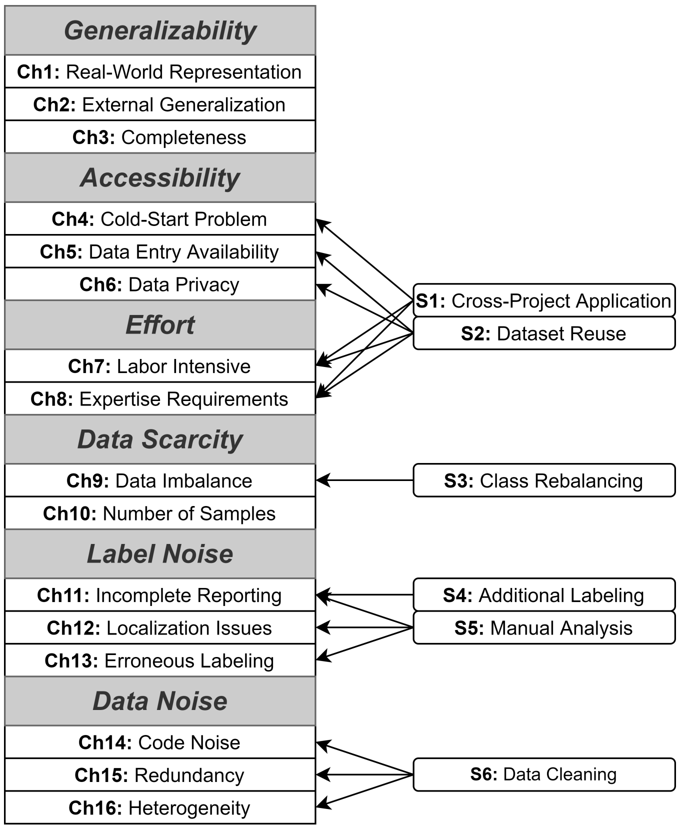

Fig. 12. A mapping of solutions onto challenges.

We provide an overview of the main solutions that have been considered for the identified data preparation challenges. As discussed in **Ch1**, the majority of the data challenges arise from the use of the real-world data; challenges for Accessibility, Effort, Data Scarcity and Label Noise are largely unique to the real-world datasets. Consequently, the majority of the data solutions are similarly aligned with the real-world data challenges. We note that not every primary study provided remediation for the reported data issues.

We also note that several of the data solutions were provided to these challenges at a model-level, rather than from the data perspective. For instance, several studies supported the use of ensemble models as they are more robust and hence less susceptible to noise and class imbalance [P42]. Furthermore, feature processing techniques, such as feature selection, were another method used to help remove data noise and increase generalizability [P22, P25, P40, P42, P47, P55]. However, we focus on the data preparation steps, so an analysis of these model-level solutions is out of the scope of this SLR.

***S1:*** *Cross-Project Application.* As discussed in **Ch4**, researchers have been motivated to overcome the cold-start problem and cost of acquiring a dataset. This has led to several efforts to apply cross-project SVP models for which a previous project's dataset is used to train an SVP model that can be applied to a new project. Due to the extreme cost of acquiring data (**Ch7&8**), the cold-start problem is thought to be further exacerbated for SVP. Hence many researchers considered cross-project SVP as an essential solution to this issue [P1, P10, P18, P21, P22, P26, P31, P37, P41, P43, P56].

Whilst this solution does solve the challenges that it seeks to address, it also adds challenges by introducing heterogeneity to the data (**Ch16**), which consequently impacts a model's performance. Furthermore, underlying problems in the original data source can still be present.

***S2:*** *Data Reuse.* With the significant challenges in the data labeling effort (**Ch7&8**), coupled with the challenges in data accessibility (**Ch5&6**), it is a more efficient choice to reuse or augment the existing datasets. This not only significantly saves time and effort in data preparation, but also enables researchers to conveniently evaluate and benchmark performance on the same datasets. As dataset reuse greatly reduces effort, it is more desirable than self-construction of a dataset. Hence, researchers have often reused the existing datasets when available; 28 out of the 61 reviewed studies reused or augmented the existing datasets in some manner.

Like **S1**, whilst this solution certainly solves the challenges regarding Accessibility and Effort, it does not address the problems in the original dataset like Generalizability, Data Scarcity, Label Noise and Data Noise. However, many researchers have seemingly assumed the validity of the prior datasets, as the studies that utilized former datasets often did not discuss other data challenges.

Data reuse is achieved through data sharing efforts of other researchers. Several researchers made their datasets publicly available for use to assist in construction of SVP models by researchers and practitioners [P1, P8, P10, P14, P16, P26, P31, P32, P36, P38, P44, P52, P56, P61].

However, the actual usability of these datasets can create further issues for this solution. Firstly, the quality and reliability of the information provided in the existing datasets is unknown and unverifiable [P4, P8, P12, P20, P28, P38, P39, P40, P44, P54]. Researchers often prefer to use a dataset about which they have complete knowledge. Researchers also build their SVP models to fit a variety of application contexts, such as specific granularities, programming languages, or SV types. Hence, several researchers found the information provided in the existing datasets insufficient to be applicable to their desired applications' contexts [P2, P4, P12, P16, P23, P27, P31, P39, P42, P43, P48, P56].

Furthermore, although many SV datasets have been created, they are not necessarily available. Researchers have reported that the existing datasets are private [P2, P16, P31] or unavailable [P2, P12, P42]. We observed that five of the shared datasets from the reviewed studies have since become unavailable due to dead links. Such problems of availability or usability of the desired datasets can lead researchers to use self-constructed datasets.

***S3:*** *Class Rebalancing.* The severe imbalance of vulnerable to non-vulnerable modules (**Ch9**) is reported by almost half of the reviewed studies (27 out of 61) as a significant data challenge. This imbalance leads to models that bias towards the majority class [50], and do not fairly consider the minority vulnerable class. As such, many studies have employed some form of class rebalancing to help remediate this issue. These rebalancing techniques fall into two main categories: Undersampling [P1, P8, P14, P15, P19, P20, P22, P30, P33, P52, P53 P55, P56, P60] and Oversampling [P8, P14, P17, P20, P36, P45, P56]. Undersampling is the process of removing samples from the majority class to match the size of the minority class, whereas oversampling duplicates or synthetically adds samples to the minority class until the size matches the majority class. Undersampling was the more popular technique used in the reviewed studies; 14 studies used undersampling in comparison to 7 that used oversampling. Researchers considered the undersampling technique to be more standard [P22, P55], effective [P30, P60] or efficient [P19, P30, P60], in comparison to oversampling. However, we note that undersampling may not always be desirable as it reduces the overall amount of data, which can have significant impacts for SV data due to the low number of samples in the minority class (**Ch10**). Oversampling does not suffer from information loss, but adds redundancy to training data (**Ch15**), which can lead to overfitting.

This solution is largely considered a necessity, as the positive impacts of class rebalancing have been widely agreed upon by the community. Furthermore, several studies have explicitly demonstrated the performance increase of models when trained on balanced data in comparison to imbalanced data [P7, P54, P60].

Alternatively, some researchers artificially constructed both their training and test datasets to be balanced [P61]. Balancing the test set like this artificially inflates the model performance, however, as the incoming real-world data will not be balanced [P1, P15]. This is a significant weakness of synthetic datasets that is expected to impact their real-world generalization, as they are constructed to over-represent the vulnerable class.

***S4:*** *Manual Analysis.* The poor labeling provided from bug reports can add a lot of label noise and inaccuracies in the data. To combat this, researchers have often assisted the labeling process through manual analysis. Over 37% of the reviewed studies (23 out of 61) assisted their data collection process with manual inspection. This allowed researchers to manually spot sources of label noise [P1, P2, P4, P7, P8, P10, P11, P14, P25, P26, P28, P29, P30, P31, P32, P42, P44, P35, P46], or made localization to code modules more accurate [P1, P2, P4, P7, P10, P14, P26, P31, P44, P45, P52]. It also helped researchers to obtain an understanding of the quality of their dataset [P13, P16, P32, P42].

Whilst manual labeling assistance can greatly improve the quality of a dataset, manual analysis is problematic in itself. It is highly effort intensive (**Ch7**), difficult (**Ch8**) and error-prone (**Ch13**).

***S5:*** *Additional labeling.* Manual analysis can only resolve issues in the vulnerable class, as the use of bug reports only provides a source of labels for this class. In reality, however, dormant or latent vulnerabilities can introduce unverifiable inconsistencies in the non-vulnerable class (**Ch11**). Hence, several researchers have used automated methods to obtain additional labels. One such approach was to use static analysis tools to label modules, in an attempt to uncover latent vulnerabilities [P37, P46]. However, this process added considerable noise to the labels in itself [P2, P46]. Li et al. [P61] used a rule-based approach to obtain the non-vulnerable modules, by filtering out functions which may have security relevance. This decreased the chance of having latent vulnerabilities in the non-vulnerable set. As these additional labeling methods often introduce their own assumptions or inaccuracies, their effectiveness is unclear.

***S6:*** *Data Cleaning.* The data noise problems can be addressed through data cleaning of the code modules. Table 5 from Section 4 lists the common data cleaning techniques for SV modules. Although the real world code is usually much noisier than the synthetic code due to inconsistent coding styles [P28, P34, P38], cleaning can still be necessary for the synthetic data sources depending on the method used to construct the vulnerable examples. Fang et al. [P50] found a large proportion of coding errors in the synthetic SARD and SQLI-Labs synthetic datasets that they manually remediated.

Data cleaning is one of the four data preparation steps [18], and consequently this solution has been largely considered a necessity. For example, Zheng et al. [P28] showed that replacing user defined strings with generic tokens improves a model's performance.

## 7 RECOMMENDATIONS

We first present an overview of the findings to our three RQs in Figure 13, to help readers to better interpret our analysis. Figure 13 displays the considerations, challenges and current solutions for the data preparation steps of SVP. This overview can help researchers or practitioners to construct SVP datasets, and be aware of the challenges and potential data quality issues.

The validity of SVP research largely relies on the quality of the data used to construct an SVP model. However, as identified in Section 5, researchers have faced a plethora of

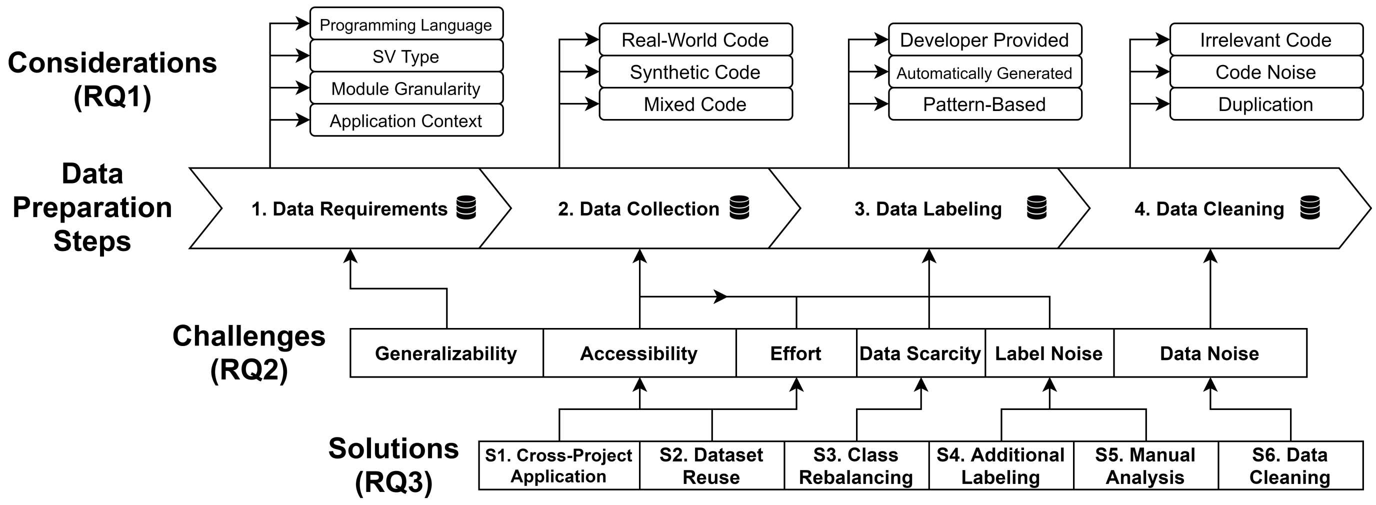

Fig. 13. An overview of the SVP considerations, challenges and solutions for data preparation.

data preparation challenges that may lead to serious pitfalls to be avoided while training SVP models. Several previous studies have called for a need of a *gold-standard* dataset [4], [13], [16]; one that overcomes the identified data preparation challenges.

We have identified several areas of the data preparation solutions in Section 6 that researchers have utilized to help address the data preparation challenges. Whilst we recommend that researchers and practitioners also adopt these solutions as they provide an initial foundation for data quality, these solutions are, by no means, complete to produce a gold-standard dataset. Particularly, many challenges for Generalizability remain unaddressed, and the solutions for Label Noise are insufficient. Manual analysis (**S5**) is often infeasible or inaccurate, and additional labeling (**S6**) can add additional noise or inconsistencies into a dataset.

Furthermore, SVP research has operated at a variety of different contexts, which makes the use of a standardized dataset difficult. Different researchers may favor different levels of granularity or applications domains. Hence, rather than proposing the guidelines for developing a gold-standard dataset, we instead propose a few recommendations for the future research directions that will help advance the SV data preparation processes and address the identified data challenges. These recommendations have been based on our review of the primary studies and observations from the supporting literature. We note that we only provide recommendations at the data-level, not for the model construction or evaluation processes that may also be needed for addressing the data challenges, as this is out of the scope of our study.

## 7.1 Improvement of the existing practices

The general SVP preparation practices are fairly consistently used in the reviewed studies. However, there is significant room for the improvement of these practices due to the many data challenges that stem from them.

**R1. Better labeling of real-world data.** We observed in Section 6 that the majority of data challenges arose from the labeling of SV data from the real-world data sources (Accessibility, Effort, Data Scarcity, Label Noise). Although the relative scarcity of SVs in the real-world data is difficult to resolve without synthetic means, much improvement can be done in regards to label noise. The major challenges for the real-world data labeling have come from the use of bug reports as a data label source. Bug report improvement through automated means is an active field of research, solving problems such as fault localization [60], and bug classification [61]. The application of these techniques for data preparation may potentially help resolve challenges in the incomplete reporting (**Ch11**) and localization issues (**Ch12**). Better automated methods for this data extraction will also greatly reduce the effort requirements (**Ch7**).

**R2. More realistic synthetic datasets.** The majority of data challenges come from the use of the real-world data, which is predominantly the preferred dataset type due to the lack of the real-world generalization of synthetic datasets (**Ch1**). If the process in which the synthetic data sources are developed is improved to produce more realistic SV examples that are more representative of real-world scenarios, then these datasets will greatly increase in value. Bug seeding is a relevant research field that intends to produce additional bug examples for the data hungry evaluation of the defect discovery methods [62]. These additional bugs are generally created through use of the existing examples. This research has recently begun to investigate the creation of more realistic bugs [63], [64]; the application and advancement of this research to SV data are a potentially promising direction.

## 7.2 Further investigation into the challenges and solutions

To overcome or provide better solutions to the data challenges, there is a need for further investigation. Generalizability appears to be an unaddressed challenge, so we outline three recommendation areas to help improve representation (**R3-5**). Many of the current solutions are also incomplete, and hence, need better advancement.

**R3. Use of more diverse language datasets.** We observe in Figure 6 that researchers have primarily constructed datasets for C/C++, PHP and Java. Whilst these are popular languages, the datasets (and hence SVP models) are very limited for other languages. This makes the application of SVP difficult to projects for these other languages, due to lack of existing data and exploration. Furthermore, the cross-language datasets are currently under-investigated, but this will require corresponding cross-language or language-agnostic techniques [P53].

**R4. Use of datasets applicable to SMEs.** The current research evaluated using real-world data has been using large reputable mature systems, as it is considered more generalizable [P23, P36, P40, P55]. However, this may be considered a limitation of the current methods when applied to Small to Medium Enterprises (SMEs). This is an important gap in the literature as it is expected that SVP would be more challenging in this context due to a lower number of SV examples (**Ch10**), poorer coding standards (**Ch14**), and less complete reporting practices (**Ch11**).

**R5. Investigation of vulnerabilities from different development lifecycle stages.** A large flaw in the use of security advisories to locate the real-world vulnerabilities is that it provides a narrow view of the different development stages; they only detail vulnerabilities identified in deployed systems. Bug reports can detail both pre-release (identified during testing) and post-release (identified in a deployed system) vulnerabilities [P5, P11]. These two SV types are drastically different in nature due to the differences in how they are detected and the impacts that they have. However, not much differentiation has been made between them. Furthermore, more simplistic vulnerabilities that can be detected earlier in the software development lifecycle, during implementation or code review, are obscured from analysis. If SVP models are intended as a form of early SV detection [21], then the use of these SVs is perhaps fundamentally flawed. The differences and impacts on the SVP process by these disparate SV types will possibly be an interesting future research direction.

**R6. More consideration of the negative class.** Fully supervised learning for SVP requires a positive class (vulnerable modules) and a negative class (non-vulnerable modules). The quality of both of these classes is highly important to ensure that a model is able to learn the appropriate patterns in the data. Whilst much efforts and manual inspection have been expended to ensure the quality of the vulnerable labels [P1, P2, P4, P7, P8, P10, P11, P14, P25, P26, P28, P29, P30, P31, P32, P42, P44, P35, P46], equal efforts have not been expended for the non-vulnerable labels. There are several different methods researchers can use to obtain the negative class: 1) they can treat the unlabeled files as the negative class, but this assumption may not be valid and careful consideration needs to be taken for versioning; 2) they can take the patches used to fix SVs as the negative class, but this creates a balanced dataset with little separation between the classes, as each SV usually has only one patch, and patches are usually small in comparison to overall code module; or 3) they can use a heuristic to help define the non-vulnerable class and avoid label noise, but this may artificially increase the separation between classes. However, despite these considerations, the negative class is usually an afterthought. Due to incomplete reporting (**Ch11**), we observed that there can be considerable inaccuracies in the non-vulnerable class. This is perhaps because researchershave only been able to obtain a data source for the vulnerable labels (i.e., the bug reports), but there is no source of labels for non-vulnerable data. Hence, we may even see a relaxation of the labeling and the fully-supervised learning requirements in the future. PU learning can be applied, in which the non-vulnerable class is instead treated as unlabeled [65].

**R7. Data quality assessment criteria.** We observed in **Ch13** that the quality of most SVP datasets is unknown. However, it is vital that researchers and practitioners are able to determine the quality of their datasets, so that they can make informed decisions about the validity and reliability of the constructed SVP models. Hence, data quality assessment criteria ought to be developed to assess SV datasets. Whilst data quality assessment has ben a long-standing practice [38], [39], these existing criteria are not relevant or applicable to the SVP domain. Our categorisation of SV data challenges in Table 6 will potentially help towards producing relevant data quality assessment criteria.

## 7.3 Consideration of other data dimensions

Finally, we observed several additional data quality dimensions that have been under-explored for this domain. Hence, we recommend that those should be better considered and investigated.

**R8. Investigation into data integration.** Data integration is the practice of merging disparate data sources into a single dataset [66]. In practice, researchers have regularly merged datasets in the reviewed primary studies to increase the amount of data. This typically involves mixing data from different projects, or between the real-world and synthetic entries. Another source of integration was even the mixing of different SV types; some researchers did not restrict SV types examined, whereas others did. However, little investigation has been done on the impact of this data mixing, and hence more research is needed on the integration techniques and the applicable data sources. For instance, Saccente et al. [P34] found that an SVP model trained on all SV types concurrently produced unreliable predictions, so they treated the dataset of each SV type separately. Researchers can even consider integrating SV related data from other common Open Source Intelligence (OSINT) sources, such as Stack Overflow [67], to provide more information to SVP models.

**R9. Consideration of data security and malicious data.** Adversarial attacks are becoming an increasingly common consideration for learning-based systems [68]. At a high-level, adversarial attacks involve the use of malicious training data or inputs with the purpose of intentionally 'fooling' a model. SVP is a learning-based process, making it susceptible to adversarial attacks; for example, adversarial inputs can be crafted to fool a model into classifying a vulnerable module as a non-vulnerable one. Despite this, no consideration of these weaknesses has been made to date, perhaps because the training data of an SVP model is usually expected to come from within a project, where the data is assumed to be secure and non-malicious. This assumption is not always valid; there is a possibility of insider threat or users who may even accidentally provide malicious data by unintentionally creating vulnerable code modules that may go undetected [69]. Similarly, the bug reports used to infer the labels of the data can often be filed by users, who may be trying to maliciously mislabel certain code modules. Furthermore, with the frequent use of cross-project, synthetic or reused datasets, data governance may not always be available. Hence, researchers ought to have more consideration and countermeasures for data security. Whilst several solutions for this issue have existed at the model-level, such as input perturbation [68], practitioners ought to ensure that their datasets come from trustworthy sources.

**R10. Consideration of timeliness.** Timeliness describes the temporal aspects of data [39]. There are two main components for timeliness. The first is the currency of data; the age of data in use compared to when it was collected. Practitioners ought to generally avoid using out-of-date data as it can lose relevancy to contemporary settings. This is particularly an issue for SV data, as vulnerabilities take time to be discovered (**Ch11**), so more complete information is usually obtainable the later data collection is performed. For instance, 10 vulnerabilities may have been reported for a codebase after three months, but an additional 10 vulnerabilities may have been reported one year later. It is unclear whether this is a significant issue in the primary studies. SV data have been generally collected at the time of analysis when self-constructing datasets. For data reuse, where this problem usually manifests, datasets are not very old as the SVP research domain is still relatively new. The oldest shared real-world dataset is the dataset from Walden et al. [P1] that was collected in 2013. However, it is still beneficial if efforts are made in future to maintain and update the existing datasets for reuse.

The second aspect of timeliness relates to the temporal nature of the data accumulation; historical artefacts have been accumulated incrementally over the lifecycle of a project. This poses an issue for SV data due to concept drift [70]; the vulnerability data and patterns change over time with the emergence of new concepts [71]. Failure to account for this temporal nature, that is, preserving data order for model training and validation, has been shown to produce unreliable models and impact the real-world generalizability [72], [73]. However, only 13% of the reviewed studies (8 out of 61) considered a time-based ordering of the data. Hence, more consideration of this issue is needed in this domain, to produce more reliable SVP models.

**R11. Consideration of understandability.** Understandability describes the ability to comprehend data [38]. Exploratory data analysis is an integral part of data science, as it helps practitioners to understand data quality issues, imbalances, and relationships within the data. This is particularly important for ML as it allows practitioners to identify the necessary data cleaning practices and to conduct feature engineering [74]. SV data can be complex due to the intricacies of the code and security weaknesses. Despite this, not much effort has been reported by researchers into data understanding and visualization.

Data visualization is one of the most powerful techniques for data understandability [74], but only a few of the reviewed studies did visual exploration of their data. Neuhaus et al. [P13] visualized the locations of SVs in a codebase and observed that the distribution of SVs are scarce and irregular. This motivated their search for specific code patterns that can be used to describe the irregular distribution. Chowdhury and Zulkerine [P19] used data visualization to identify that files are unlikely to contain SVs in multiple different versions, and hence, determined that vulnerability history of a module may be a poor indicator. The future efforts in data understandability for SV datasets, using similar techniques to the ones aforementioned, may be able to yield novel impactful insights for this data, and assist in SVP model creation. Better data understandability can also help practitioners who lack data science expertise with setting up SVP models.

**R12. Increased data trustworthiness.** Although data reuse has been popular for SVP (**S2**), we observed that there are many issues regarding the availability of these datasets. Furthermore, the incomplete reporting practices of SV data has led to a lack of trustworthiness; the integrity of the datasets is not verifiable. The reporting of the datasets has often been insufficient to allow for proper replication, making the datasets hard to validate. For instance, many of the reviewed studies did not report the version of the data source or specific extraction steps. Furthermore, most data preparation processes used extensive manual inspection or labeling (**S5**), which is hard to replicate due to its subjectivity. Although some studies attempted to remediate this by making their datasets available, not many of them made the code and methods used to create these datasets available, which has made reproduction and adaptation efforts very difficult. Riom et al. [75] found the replication of a seminal work [P16] infeasible due to these challenges.

The underlying problem of the poor data trustworthiness is a lack of proper data reporting and storage efforts. Researchers ought to make more efforts in the future to specify the exact details and processes of data preparation, to allow for easy reuse, understanding and augmentation of the datasets. One such example process is called *Datasheets for Datasets*, proposed by Gebru et al. [76]. Such documentation will also help practitioners to better understand the capabilities and implications of the produced SVP models, since the models are highly reliant on the dataset used to produce them.

Another potential solution is open data sharing platforms or repositories for enhancing data availability. Such platforms can also provide vital information regarding data provenance and quality. These efforts will potentially also encourage data maintenance, which is another issue that we have discussed in **R10**. Considerable efforts towards such a platform have already been made in the Software Engineering domain through the PROMISE repository [77] and the SEACRAFT repository [78]. However, these efforts still currently fail to ensure data provenance and maintenance.

## 8 Threats to Validity

This review has been designed and executed by carefully following the guidelines for SLR provided by Kitchenham et al. [27]. Hence, we have identified the potential validity threats to this SLR and taken appropriate steps to minimize the expected impact of the identified threats as per the SLR guidelines. We discuss the validity threats considered for this SLR below.

A standard threat to any SLR is selection bias; some relevant papers may be missed during the selection process of the SLR. Whilst there is a possibility that our search and study selection process may have missed some relevant papers, we systematically drove the paper selection process by following the SLR guidelines and the recommended practices to minimize such possibility of missing the relevant papers. For example, we chose a meta search engine, SCOPUS, that indexes all the well-known computer science and software engineering digital libraries such as IEEE, ACM, Elsevier, and Springer. Furthermore, we iteratively refined our search string using the quasi-gold sensitivity approach defined by Zhang et al. [28] until we were confident that our search string had retrieved the majority of the key papers in this research area. Finally, we used backward and forward snowballing to help capture any studies that might have been missed during our automatic search. To avoid the study selection bias by the authors, we initially conducted a collaborative pilot study selection on 100 papers to ensure consistency. Furthermore, any paper that an individual author was not confident about including/excluding was discussed between the first two authors before making a final decision.

Additional validity threats can be introduced through the quality assessment, data extraction and thematic analysis processes of this SLR. Inaccuracies in these processes can be introduced by human-error and researcher-bias. The first two authors jointly carried out the pilot activities for these processes to help ensure author consistency. Furthermore, all the data-extraction activities were cross-checked by the authors, with disagreements resolved through discussions.

Finally, our results may be affected by publication bias; research is biased towards the publication of positive results over negative results [27]. Hence, it is expected that researchers would also be reluctant to report the major (data) limitations if it is not the focus of a study. As our findings were grounded in the data extracted from the primary studies, we were only able to report the considerations and challenges explicitly discussed in the papers. Hence, our findings may not be exhaustive. However, we assume that our findings were able to capture the major data challenges and considerations, due to the quality and integrity of our selected primary studies which we ensured through our inclusion/exclusion and data quality assessment criteria from Section 3.2.

## 9 Conclusion

This paper reports our research effort aimed at systematically reviewing the literature on the data preparation processes for SVP. Data challenges serve as the major barrier for research and industrial adoption, hence, our study aims to shed light on these important issues to help advance this emerging research field. We have conducted rigorous analysis and systematic synthesis of 61 papers reporting research on SVP.

The main aim of this research was to identify the considerations, challenges and solutions of data preparation for SVP, by answering three research questions. Firstly, we have identified the major decisions made by researchers for the data preparation processes, to help inform the state of the practice. Secondly, through our thematic analysis, we have derived a taxonomy of 16 identified data challenges that researchers face for constructing data-driven methods for SVP. These challenges involve the data Generalizability, Accessibility, label collection Effort and Scarcity, and both Label and Data Noise. Thirdly, we similarly categorized and mapped the data solutions that researchers utilize. However, these solutions do not address all sorts of data preparation challenges identified by this SLR.

We have found that challenges are particularly pertinent for the data labelling process that has consequently attracted the majority of the identified solutions. Due to these significant challenges, data reuse is a common solution to reduce data construction effort and difficulties. Alternatively, the use of synthetic datasets is an attractive option as these datasets artificially solve challenges for Accessibility, Effort, Data Scarcity and Label Noise, but these datasets experience significant challenges with Generalizability, which severely limits their value.

The findings of our SLR aim to help researchers and practitioners to understand the key SV data preparation considerations and challenges; such evidence-based understanding can aid future SVP research and practice. By consolidating the state of the practice into an integrated source of information, our study purports to assist practitioners in improving their data preparation practices for building and deploying SVP models. Furthermore, the taxonomy of the data preparation challenges developed based on this SLR can be used to identify and classify the data challenges that practitioners may encounter; and our categorization of the identified solutions is expected to help identify the best practices of data preparation for SVP models. Such improvements are expected in turn to improve the quality of SVP models, as their efficacy hinges on the data quality; *Garbage In, Garbage Out*.

The findings also help inform the Software Engineering research community of the main limitations and barriers of adoption for SVP, and to help direct the future research in this area. We have provided guidance for the areas of investigation through our 12 recommendations. Whilst we acknowledge that our study does not provide a complete view of the SVP process on its own, the data is undoubtedly one of the most important components for any data-driven process in, and hence, we take the first step to highlight this area of research. By taking steps toward improving data preparation and data quality, we expect to enable the creation of more reliable and trustworthy SVP research and help make automated software security analytics a reality.

## Acknowledgments

This work has been supported by the Cyber Security Cooperative Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centre Programme.

## Appendix
## Primary Studies

P1 J. Walden, J. Stuckman and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," *International Symposium on Software Reliability Engineering*, 2014.

P2 Y. Zhou, S. Liu, J. Siow, X. Du and Y. Liu, ”Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks,” *Conference on Neural Information Processing Systems*, 2019.

P3 J. Tian, W. Xing and Z. Li, ”BVDetector: A program slice-based binary code vulnerability intelligent detection system,” *Information and Software Technology*, 2020.

P4 C. Theisen and L. Williams, ”Better together: Comparing vulnerability prediction models,” *Information and Software Technology*, 2020.

P5 P. Morrison, K. Herzig, B. Murphy and L. Williams, ”Challenges with applying vulnerability prediction models,” *Symposium and Bootcamp on the Science of Security*, 2015.

P6 A. Fidalgo, I. Medeiros, P. Antunes and N. Neves, ”Towards a Deep Learning Model for Vulnerability Detection on Web Application Variants,” *International Conference on Software Testing, Verification and Validation Workshops*, 2020.

P7 X. Ban, S. Liu, C. Chen and C. Chua, ”A performance evaluation of deep-learnt features for software vulnerability detection,” *Concurrency and Computation: Practice and Experience*, 2019.

P8 Z. Li, D. Zou, J. Tang, Z. Zhang, M. Sun and H. Jin, ”A comparative study of deep learning-based vulnerability detection system,” *IEEE Access*, 2019.

P9 T. Nguyen, T. Le, K. Nguyen, O. de Vel, P. Montague, J. Grundy and D. Phung, ”Deep Cost-Sensitive Kernel Machine for Binary Software Vulnerability Detection,” *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020.

P10 G. Lin, J. Zhang, W. Luo, L. Pan, O. D. Vel, P. Montague and Y. Xiang, ”Software Vulnerability Discovery via Learning Multi-domain Knowledge Bases,” *IEEE Transactions on Dependable and Secure Computing*, 2019.

P11 M. Gegick, L. Williams, J. Osborne and M. Vouk, ”Prioritizing software security fortification through code-level metrics,” *ACM Workshop on Quality of protection*, 2008.

P12 M. Jimenez, M. Papadakis and Y. L. Traon, ”Vulnerability prediction models: A case study on the Linux Kernel,” *International Working Conference on Source Code Analysis and Manipulation*, 2016.

P13 S. Neuhaus, T. Zimmermann, C. Holler and A. Zeller, ”Predicting vulnerable software components,” *ACM Conference on Computer and Communications Security*, 2007.

P14 S. Liu, G. Lin, Q. Han, S. Wen, J. Zhang and Y. Xiang, ”DeepBalance: Deep-Learning and Fuzzy Oversampling for Vulnerability Detection,” *IEEE Transactions on Fuzzy Systems*, 2020.

P15 M. Zagane, M. K. Abdi and M. Alenezi, ”Deep Learning for Software Vulnerabilities Detection Using Code Metrics,” *IEEE Access*, 2020.

P16 H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl and Y. Acar, ”VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits,” *ACM SIGSAC Conference on Computer and Communications Security*, 2015.

P17 Y. Joonseok, R. Duksan and B. Jongmoon, ”Improving vulnerability prediction accuracy with Secure Coding Standard violation measures,” *International Conference on Big Data and Smart Computing (BigComp)*, 2016.

P18 V. Nguyen, T. Le, T. Le, K. Nguyen, O. De Vel, P. Montague, L. Qu and D. Phung, ”Deep Domain Adaptation for Vulnerable Code Function Identification,” *International Joint Conference on Neural Networks*, 2019.

P19 I. Chowdhury and M. Zulkernine, ”Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities,” *Journal of Systems Architecture*, 2011.

P20 H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy and A. Ghose, ”Automatic Feature Learning for Predicting Vulnerable Software Components,” *IEEE Transactions on Software Engineering*, 2018.

P21 A. Meneely and L. Williams, ”Strengthening the empirical analysis of the relationship between Linus’ Law and software security,” *International Symposium on Empirical Software Engineering and Measurement*, 2010.

P22 X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu and Z. Wang, ”Large-Scale Empirical Studies on Effort-Aware Security Vulnerability Prediction Methods,” *IEEE Transactions on Reliability*, 2020.

P23 X. Cheng, H. Wang, J. Hua, M. Zhang, G. Xu, L. Yi and Y. Sui, ”Static detection of control-flow-related vulnerabilities using graph embedding,” *International Conference on Engineering of Complex Computer Systems*, 2019.

P24 V. H. Nguyen and L. M. S. Tran, ”Predicting vulnerable software components with dependency graphs,” *International Workshop on Security Measurements and Metrics*, 2010.

P25 L. K. Shar and H. B. K. Tan, ”Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns,” *Information and Software Technology*, 2013.

P26 S. Liu, G. Lin, L. Qu, J. Zhang, O. D. Vel, P. Montague and Y. Xiang, ”CD-VulD: Cross-Domain Vulnerability Discovery based on Deep Domain Adaptation,” *IEEE Transactions on Dependable and Secure Computing*, 2020.

P27 B. M. Padmanabhuni and H. B. K. Tan, ”Buffer Overflow Vulnerability Prediction from x86 Executables Using Static Analysis and Machine Learning,” *Annual Computer Software and Applications Conference*, 2015.

P28 W. Zheng, J. Gao, X. Wu, F. Liu, Y. Xun, G. Liu and X. Chen, ”The impact factors on the performance of machine learning-based vulnerability detection: A comparative study,” *Journal of Systems and Software*, 2020.

P29 H. Wang, G. Ye, Z. Tang, S. H. Tan, S. Huang, D. Fang, Y. Feng, L. Bian and Z. Wang, ”Combining Graph-Based Learning With Automated Data Collection for Code Vulnerability Detection,” *IEEE Transactions on Information Forensics and Security*, 2021.

P30 Y. Shin, A. Meneely, L. Williams and J. A. Osborne, ”Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities,” *IEEE Transactions on Software Engineering*, 2011.

P31 G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, O. De Vel and P. Montague, ”Cross-Project Transfer Representation Learning for Vulnerable Function Discovery,” *IEEE Transactions on Industrial Informatics*, 2018.

P32 Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng and Y. Zhong, ”Vuldeepecker: A deep learning-based system for vulnerability detection,” *Usenix Network and Distributed System Security Symposium*, 2018.

P33 Y. Shin and L. Williams, ”Can traditional fault prediction models be used for vulnerability prediction?,” *Empirical Software Engineering*, 2013.

P34 N. Saccente, J. Dehlinger, L. Deng, S. Chakraborty and Y. Xiong, ”Project achilles: A prototype tool for static method-level vulnerability detection of Java source code using a recurrent neural network,” *International Conference on Automated Software Engineering Workshop*, 2019.

P35 Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun and S. Li, ”Combining Software Metrics and Text Features for Vulnerable File Prediction,” *International Conference on Engineering of Complex Computer Systems*, 2016.

P36 M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. Le Traon and M. Harman, ”The importance of accounting for real-world labeling when predicting software vulnerabilities,” *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

P37 S. Moshtari, A. Sami and M. Azimi, ”Using complexity metrics to improve software security,” *Computer Fraud, Security*, 2013.

P38 D. Zou, S. Wang, S. Xu, Z. Li and H. Jin, ”$\mu$VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection,” *IEEE Transactions on Dependable and Secure Computing*, 2019.

P39 S. Ghaffarian and H. Shahriari, ”Neural software vulnerability analysis using rich intermediate graph representations of programs,” *Information Sciences*, 2021.

P40 M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias and D. Tzovaras, ”Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises,” *Enterprise Information Systems*, 2020.

P41 I. Abunadi and M. Alenezi, ”An empirical investigation of security vulnerabilities within web applications,” *Journal of Universal Computer Science*, 2016.

P42 L. Yang, X. Li and Y. Yu, ”VulDigger: A Just-in-Time and Cost-Aware Tool for Digging Vulnerability-Contributing Changes,” *IEEE Global Communications Conference*, 2017.

P43 V. Nguyen, T. Le, O. de Vel, P. Montague, J. Grundy and D. Phung, ”Dual-Component Deep Domain Adaptation: A New Approach for Cross Project Software Vulnerability Detection,” *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020.

P44 Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu and Z. Chen, ”SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities,” *IEEE Transactions on Dependable and Secure Computing*, 2021.

P45 S. Liu, M. Dibaei, Y. Tai, C. Chen, J. Zhang and Y. Xiang, ”Cyber Vulnerability Intelligence for Internet of Things Binary,” *IEEE Transactions on Industrial Informatics*, 2020.

P46 R. Scandariato, J. Walden, A. Hovsepyan and W. Joosen, ”Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, 2014.

P47 N. Medeiros, N. Ivaki, P. Costa and M. Vieira, "Software metrics as indicators of security vulnerabilities," *Software Reliability Engineering*, 2017.

P48 M. A. Albahar, "A Modified Maximal Divergence Sequential Auto-Encoder and Time Delay Neural Network Models for Vulnerable Binary Codes Detection," *IEEE Access*, 2020.

P49 X. Duan, J. Wu, S. Ji, Z. Rui, T. Luo, M. Yang and Y. Wu, "Vulsniper: Focus your attention to shoot fine-grained vulnerabilities," *International Joint Conference on Artificial Intelligence*, 2019.

P50 Y. Fang, S. Han, C. Huang and R. Wu, "TAP: A static analysis model for PHP vulnerabilities based on token and deep learning technology," *PloS one*, 2019.

P51 D. Cao, J. Huang, X. Zhang and X. Liu, "FTCLNet: Convolutional LSTM with Fourier Transform for Vulnerability Detection," *Trust, Security and Privacy in Computing and Communications*, 2020.

P52 Z. Yu, C. Theisen, L. Williams and T. Menzies, "Improving Vulnerability Inspection Efficiency Using Active Learning," *IEEE Transactions on Software Engineering*, 2019.

P53 A. Figueiredo, T. Lide, D. Matos and M. Correia, "MERLIN: Multi-Language Web Vulnerability Detection," *International Symposium on Network Computing and Applications*, 2020.

P54 Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Comak and L. Karacay, "Vulnerability Prediction from Source Code Using Machine Learning," *IEEE Access*, 2020.

P55 X. Chen, Z. Yuan, Z. Cui, D. Zhang and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," *IET Software*, 2020.

P56 J. Stuckman, J. Walden and R. Scandariato, "The Effect of Dimensionality Reduction on Software Vulnerability Prediction Models," *IEEE Transactions on Reliability*, 2017.

P57 Y. Tang, F. Zhao, Y. Yang, H. Lu, Y. Zhou and B. Xu, "Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-Aware Perspective," *IEEE International Conference on Software Quality, Reliability and Security*, 2015.

P58 J. Domanska, M. Siavvas and E. Gelenbe, "Efficient Feature Selection for Static Analysis Vulnerability Prediction," *K. Filus, P. Boryszko, Sensors*, 2021.

P59 Y. Shin and L. Williams, "An initial study on the use of execution complexity metrics as indicators of software vulnerabilities," *International Workshop on Software Engineering for Secure Systems*, 2011.

P60 N. Medeiros, N. Ivaki, P. Costa and M. Vieira, "Vulnerable Code Detection Using Software Metrics and Machine Learning," *IEEE Access*, 2020.

P61 R. Li, C. Feng, X. Zhang and C. Tang, "A Lightweight Assisted Vulnerability Discovery Method Using Deep Neural Networks," *IEEE Access*, 2019.

## REFERENCES

[1] R. Sobers, "134 cybersecurity statistics and trends for 2021," 2021. [Online]. Available: https://www.varonis.com/blog/cybersecurity-statistics/

[2] H. Shahriar and M. Zulkernine, "Mitigating program security vulnerabilities: Approaches and challenges," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, pp. 1–46, 2012.

[3] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, vol. 18, no. 1, pp. 25–59, 2013.

[4] H. Hanif, M. H. N. M. Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, p. 103009, 2021.

[5] D. Pyle, *Data preparation for data mining*. morgan kaufmann, 1999.

[6] A. Zheng and A. Casari, *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[7] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *2014 IEEE 25th international symposium on software reliability engineering*. IEEE, 2014, pp. 23–33.

[8] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 421–428.

[9] A. Anwar, A. Abusnaina, S. Chen, F. Li, and D. Mohaisen, "Cleaning the nvd: Comprehensive quality assessment, improvements, and analyses," *arXiv preprint arXiv:2006.15074*, 2020.

[10] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang, "Code analysis for intelligent cyber systems: A data-driven approach," *Information sciences*, vol. 524, pp. 46–58, 2020.

[11] M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. Le Traon, and M. Harman, "The importance of accounting for real-world labelling when predicting software vulnerabilities," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 695–705.

[12] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 812–823.

[13] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: a survey," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1825–1848, 2020.

[14] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[15] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, 2015, pp. 1–9.

[16] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang, "Software vulnerability analysis and discovery using deep learning techniques: A survey," *IEEE Access*, 2020.

[17] A. O. A. Semasaba, W. Zheng, X. Wu, and S. A. Agyemang, "Literature survey of deep learning-based vulnerability analysis on source code," *IET Software*, 2020.

[18] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[19] "The state of ai and machine learning," Appen, Tech. Rep., 2019.

[20] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 529–540.

[21] R. Croft, D. Newlands, Z. Chen, and M. A. Babar, "An empirical study of rule-based and learning-based approaches for static application security testing," *arXiv preprint arXiv:2107.01921*, 2021.

[22] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 1–36, 2017.

[23] C. Theisen and L. Williams, "Better together: Comparing vulnerability prediction models," *Information and Software Technology*, vol. 119, p. 106204, 2020.

[24] Z. Li and Y. Shao, "A survey of feature selection for vulnerability prediction using feature-based machine learning," in *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*, 2019, pp. 36–42.

[25] T. H. Le, H. Chen, and M. A. Babar, "A survey on data-driven software vulnerability assessment and prioritization," *arXiv preprint arXiv:2107.08364*, 2021.

[26] S. K. Singh and A. Chaturvedi, "Applying deep learning for discovery and analysis of software vulnerabilities: A brief survey," *Soft Computing: Theories and Applications*, pp. 649–658, 2020.

[27] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[28] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology*, vol. 53, no. 6, pp. 625–637, 2011.

[29] C. Schardt, M. B. Adams, T. Owens, S. Keitz, and P. Fontelo, "Utilization of the pico framework to improve searching pubmed for clinical questions," *BMC medical informatics and decision making*, vol. 7, no. 1, pp. 1–6, 2007.

[30] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111–147, 2017.

[31] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, p. 106287, 2020.

[32] B. Sabir, F. Ullah, M. A. Babar, and R. Gaire, "Machine learning for detecting data exfiltration: A review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–47, 2021.

[33] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[34] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[35] V. Garousi and M. Felderer, "Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 170–179.

[36] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[37] QSR International, "Nvivo." [Online]. Available: https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home

[38] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data quality assessment," *Communications of the ACM*, vol. 45, no. 4, pp. 211–218, 2002.

[39] F. Sidi, P. H. S. Panahy, L. S. Affendey, M. A. Jabar, H. Ibrahim, and A. Mustapha, "Data quality: A survey of data quality dimensions," in *2012 International Conference on Information Retrieval & Knowledge Management*. IEEE, 2012, pp. 300–304.

[40] "Github." [Online]. Available: https://github.com/

[41] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Transactions on Software Engineering*, 2021.

[42] NIST, "Software assurance and reference dataset (sard)." [Online]. Available: https://samate.nist.gov/SARD/testsuite.php

[43] OWASP, "Owasp benchmark project." [Online]. Available: https://owasp.org/www-project-benchmark/

[44] VulnSpy, "Sqli labs." [Online]. Available: https://www.vulnspy.com/sqli-labs/

[45] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Software Engineering*, vol. 16, no. 3, pp. 365–395, 2011.

[46] NIST, "National vulnerability database." [Online]. Available: https://nvd.nist.gov/

[47] Atlassian, "Jira." [Online]. Available: https://www.atlassian.com/software/jira

[48] Mozilla, "Bugzilla." [Online]. Available: https://www.bugzilla.org/

[49] J. Hernández-González, I. Inza, and J. A. Lozano, "Weak supervision and other non-standard classification problems: a taxonomy," *Pattern Recognition Letters*, vol. 69, pp. 49–55, 2016.

[50] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[51] P. S. Kochhar, D. Wijedasa, and D. Lo, "A large scale study of multiple programming languages and code quality," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 563–573.

[52] S.-a. Knight and J. Burn, "Developing a framework for assessing information quality on the world wide web." *Informing Science*, vol. 8, 2005.

[53] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Systems with Applications*, vol. 41, no. 4, pp. 2065–2073, 2014.

[54] M. F. Bosu and S. G. MacDonell, "A taxonomy of data quality challenges in empirical software engineering," in *2013 22nd Australian Software Engineering Conference*. IEEE, 2013, pp. 97–106.

[55] M. Dowd, J. McDonald, and J. Schuh, *The art of software security assessment: Identifying and preventing software vulnerabilities*. Pearson Education, 2006.

[56] S. Barnum and G. McGraw, "Knowledge for software security," *IEEE Security & Privacy*, vol. 3, no. 2, pp. 74–78, 2005.

[57] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of management information systems*, vol. 12, no. 4, pp. 5–33, 1996.

[58] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[59] K. Herzig and A. Zeller, "The impact of tangled code changes," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 121–130.

[60] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 14–24.

[61] Y. Jiang, P. Lu, X. Su, and T. Wang, "Ltrwes: A new framework for security bug report detection," *Information and Software Technology*, vol. 124, p. 106314, 2020.

[62] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[63] J. Patra and M. Pradel, "Semantic bug seeding: A learning-based approach for creating realistic bugs," *synthesis*, vol. 15, p. 53, 2021.

[64] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "Learning how to mutate source code from bug-fixes," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 301–312.

[65] T. H. M. Le, D. Hin, R. Croft, and M. A. Babar, "Puminer: Mining security posts from developer question and answer websites with pu learning," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 350–361.

[66] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002, pp. 233–246.

[67] T. H. Le, R. Croft, D. Hin, and M. A. Babar, "A large-scale study of security vulnerability support on developer q&a websites," *arXiv preprint arXiv:2008.04176*, 2020.

[68] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[69] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–40, 2019.

[70] T. H. M. Le, B. Sabir, and M. A. Babar, "Automated software vulnerability assessment with concept drift," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 371–382.

[71] M. A. Williams, S. Dey, R. C. Barranco, S. M. Naim, M. S. Hossain, and M. Akbar, "Analyzing evolving trends of vulnerabilities in national vulnerability database," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3011–3020.

[72] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, 2017.

[73] D. Falessi, J. Huang, L. Narayana, J. F. Thai, and B. Turhan, "On the need of preserving order of data when validating within-project defect classifiers," *Empirical Software Engineering*, vol. 25, no. 6, pp. 4805–4830, 2020.

[74] T. M. Mitchell, "Machine learning and data mining," *Communications of the ACM*, vol. 42, no. 11, pp. 30–36, 1999.

[75] T. Riom, A. Sawadogo, K. Allix, T. F. Bissyandé, N. Moha, and J. Klein, "Revisiting the vccfinder approach for the identification of vulnerability-contributing commits," *Empirical Software Engineering*, vol. 26, no. 3, pp. 1–30, 2021.

[76] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford, "Datasheets for datasets," *arXiv preprint arXiv:1803.09010*, 2018.

[77] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases." School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[78] "The seacraft repository of empirical software engineering data," 2017.