

Mapping NVD Records to Their VFCs: How Hard is it?

HUU HUNG NGUYEN, Singapore Management University, Singapore

DUC MANH TRAN, Singapore Management University, Singapore

YIRAN CHENG, University of Chinese Academy of Sciences, China

THANH LE-CONG, University of Melbourne, Australia

HONG JIN KANG, University of Sydney, Australia

RATNADIRA WIDYASARI, Singapore Management University, Singapore

SHAR LWIN KHIN, Singapore Management University, Singapore

OUH ENG LIEH, Singapore Management University, Singapore

TING ZHANG*, Singapore Management University, Singapore

DAVID LO, Singapore Management University, Singapore

Mapping National Vulnerability Database (NVD) records to vulnerability-fixing commits (VFCs) is crucial for vulnerability analysis but challenging due to sparse explicit links in NVD references. This study explores this mapping's feasibility through an empirical approach. Manual analysis of NVD references showed Git references enable over 86% success, while non-Git references achieve under 14%. Using these findings, we built an automated pipeline extracting 31,942 VFCs from 20,360 NVD records (8.7% of 235,341) with 87% precision, mainly from Git references. To fill gaps, we mined six external security databases, yielding 29,254 VFCs for 18,985 records (8.1%) at 88.4% precision, and GitHub repositories, adding 3,686 VFCs for 2,795 records (1.2%) at 73% precision. Combining these, we mapped 26,710 unique records (11.3% coverage) from 7,634 projects, with overlap between NVD and external databases, plus unique GitHub contributions. Despite success with Git references, 88.7% of records remain unmapped, highlighting the difficulty without Git links. This study offers insights for enhancing vulnerability datasets and guiding future automated security research.

ACM Reference Format:

Huu Hung Nguyen, Duc Manh Tran, Yiran Cheng, Thanh Le-Cong, Hong Jin Kang, Ratnadira Widyasari, Shar Lwin Khin, Ouh Eng Lieh, Ting Zhang, and David Lo. 2025. Mapping NVD Records to Their VFCs: How Hard is it?. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 27 pages. <https://doi.org/XXXXXXXX.XXXXXXX>

*Ting Zhang is the corresponding author.

Authors' Contact Information: Huu Hung Nguyen, Singapore Management University, Singapore, huuhungn@smu.edu.sg; Duc Manh Tran, dmtran@smu.edu.sg, Singapore Management University, Singapore; Yiran Cheng, University of Chinese Academy of Sciences, China, chengyiran@iie.ac.cn; Thanh Le-Cong, tlecong@student.unimelb.edu.au, University of Melbourne, Australia; Hong Jin Kang, University of Sydney, Australia, hongjin.kang@sydney.edu.au; Ratnadira Widyasari, ratnadiraw@smu.edu.sg, Singapore Management University, Singapore; Shar Lwin Khin, Singapore Management University, Singapore, lkshar@smu.edu.sg; Ouh Eng Lieh, Singapore Management University, Singapore, elouh@smu.edu.sg; Ting Zhang, Singapore Management University, Singapore, tingzhang.2019@phdcs.smu.edu.sg; David Lo, Singapore Management University, Singapore, davidlo@smu.edu.sg.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference acronym 'XX, Woodstock, NY

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/18/06

<https://doi.org/XXXXXXXX.XXXXXXX>

1 Introduction

Open-source software (OSS) vulnerabilities are a prime target for malicious actors, often leading to severe security breaches, including data leaks and system disruptions [11, 15, 18, 33, 40]. To standardize the tracking of these vulnerabilities, the Common Vulnerabilities and Exposures (CVE) system [2], managed by MITRE, assigns unique identifiers to each disclosed vulnerability. These identifiers are further enriched by the National Institute of Standards and Technology (NIST) in the National Vulnerability Database (NVD), which provides additional details such as severity ratings and references to patches or fixes [6]. With over 200,000 records, the NVD serves as a cornerstone for vulnerability research. However, while the NVD offers a wealth of information, it often falls short of providing a complete picture of the vulnerability lifecycle, particularly in understanding how vulnerabilities are mitigated in practice.

A critical aspect of this lifecycle is the identification and analysis of vulnerability-fixing commits (VFCs), which are essential for understanding how vulnerabilities are resolved and tracing the commits that introduced them in the first place [24, 35]. VFCs, often referred to as security patches, are not only crucial for resolving vulnerabilities but also for building comprehensive datasets that can inform future research and development. While some NVD records include Git links to VFCs in their reference fields, marked with the Patch tag, such links are scarce; only 6.7% of NVD records provide this information. This scarcity highlights a significant gap in the availability of actionable data for researchers and practitioners.

The study of historical VFCs offers valuable insights into vulnerability trends, recurring patterns, and the broader context of how vulnerabilities emerge across diverse software systems [20, 28, 37]. Moreover, detailed information on VFCs is indispensable for advancing automated vulnerability detection methods, especially as deep learning and data-driven approaches become increasingly popular in security research [14, 20, 32, 36]. However, the limited availability of VFCs poses a significant challenge, hindering progress in both research and practical applications.

A critical research problem arises in mapping NVD records to their corresponding VFCs. While prior work has explored various methods for identifying VFCs [14, 20, 25, 36, 41], the focus has largely been on improving the accuracy of general VFC identification rather than specifically mapping NVD records to their fixes. Recent studies have attempted to extract VFC information from NVD records [14, 21, 32] or proposed methods to map NVD records to their VFCs [25, 36]. However, a fundamental question remains unanswered: *How hard is it to map NVD records to their corresponding VFCs comprehensively?* This question drives our research, as understanding the difficulty of this mapping is crucial to unlocking the NVD's full potential for vulnerability analysis. This research problem extends the scope of prior work by not only aiming to enhance the accuracy of VFC identification but also seeking to maximize the coverage of identified mappings. Despite its importance, there is a lack of systematic investigation into the challenges of this process. To fill this gap, our study explores the difficulty of identifying NVD-VFC mappings and provides actionable insights for future research in software security. Specifically, we address the following research questions (RQs):

- **RQ1:** What methods do humans use to identify VFCs associated with a given NVD record?
- **RQ2:** What proportion of records in the NVD include patch links in their reference sections?
- **RQ3:** How many patch links can be discovered from external security databases?
- **RQ4:** How many patch links can be identified through GitHub repository analysis?
- **RQ5:** How do the VFCs extracted from NVD references, external security databases, and GitHub repositories compare in terms of precision, coverage, and overlap?

In this paper, we present an empirical study to tackle this challenge head-on, exploring the feasibility and obstacles of linking NVD records to VFCs across their diverse categories. We

began with a manual investigation of NVD references, categorizing records by Git and Patch-tag presence to assess human identification strategies and their success rates. Building on these findings, we developed an automated pipeline to extract VFCs from NVD references, achieving high precision for Git-linked records. To address remaining gaps, we mined external security databases (e.g., Snyk, GitHub Advisory) and analyzed GitHub repositories using state-of-the-art tools like Prospector [36], comparing all sources for precision, coverage, and overlap. Our results show that while 26,710 records (11.3% of the NVD) can be mapped with 86.1% VFC precision, around 88.7% remain unmapped, particularly those without Git references, highlighting significant challenges and opportunities.

Our contributions can be summarized as follows:

- We performed a manual investigation to understand how humans identify VFCs from NVD references.
- We performed an empirical study to systematically collect VFCs from NVD references, external security databases, and GitHub repositories, using manual verification to assess precision.
- We evaluated the effectiveness of extracting VFCs from these three sources, comparing precision, coverage, and overlap to offer practical insights for future research and applications.
- We compiled a dataset of 37,441 VFCs across 26,710 NVD records, achieving an average precision of 86.1% - to our knowledge, the largest VFC dataset to date.

The rest of this article is organized as follows: Section 2 discusses the background; Section 3 presents our study design; Section 4 presents the experimental results; Section 5 elaborates on the lessons learned and threats to validity; Section 6 discusses other relevant studies; finally, Section 7 concludes our study and outlines potential future work.

2 Background and Related Work

In this section, we provide the background knowledge related to our work. First, we provide an overview of VFCs. Second, we review existing approaches to VFC mapping and their limitations. Third, we discuss the challenges associated with mapping NVD records to their VFCs. Last, we demonstrate the different NVD record categories.

2.1 Overview of Vulnerability-fixing Commits

VFCs, also known as *security patches*, are code changes that fix or mitigate vulnerabilities. VFCs play a crucial role in various vulnerability-related research areas, such as vulnerability type detection [34], vulnerability-introducing commit detection [13], and automated program repair [22]. They enable researchers to perform in-depth analyses of vulnerabilities and the corresponding code changes, providing valuable insights into how vulnerabilities are introduced and resolved.

Moreover, VFCs are essential for developing automated approaches to vulnerability detection and repair, and vulnerable OSS version identification [19, 22, 34]. Despite the NVD publicly disclosing numerous vulnerabilities, a significant portion of vulnerabilities and their corresponding VFCs remain undisclosed [27]. This gap in disclosure poses challenges for both security research and practical applications.

2.2 Existing Approaches to VFC Mapping

Most studies identify VFCs by directly utilizing Git links provided in NVD references [12, 14, 21, 29, 32]. For example, the CrossVul dataset [32] collects GitHub commits from NVD references and processes pull request references to obtain all relevant commits. Similarly, both the Big-Vul [21] and CVEFixes [14] datasets rely on reference links to Git repositories for VFCs, extending beyond GitHub to include repositories like Google Android (Big-Vul) and GitLab and Bitbucket (CVEFixes).

Other studies extract CVE and VFC mappings from external security databases. For instance, the DiverseVul dataset [17] uses databases from Snyk.io and bugzilla.redhat.com, while Sun et al. [38] leverage the GitHub Advisory Database.

Another approach involves searching GitHub repositories directly. Although this area has recently gained attention, few studies exist [23, 25, 36, 41, 42]. Notable recent methods include FixFinder [23], which preceded Prospector [36], and PatchFinder [25], which has demonstrated improved performance over earlier methods like PatchScout [39] and VCMatch [41].

MoreFixes [12], a recent dataset, builds upon Prospector [36] by extracting GitHub repository URLs from GitHub Security Advisories [4] and NVD references, then employing Prospector to identify and score potential VFCs using the repository URL and CVE-ID.

Most existing studies focus narrowly on a single aspect of NVD record mapping, failing to provide a comprehensive perspective. In contrast, our study offers the most holistic approach to date for mapping NVD records to VFCs. We begin by integrating a dedicated manual investigation to inform and refine automated extraction processes. Unlike prior works that rely solely on NVD reference links, we enrich our analysis by incorporating data from external security databases and GitHub repositories, improving the depth of VFC identification. Furthermore, our methodology stands out through multiple rounds of manual verification on sampled VFCs at each stage, ensuring high precision in identifying the relevant commits.

2.3 Challenges in Mapping NVD Records to their VFCs

Despite the importance of VFCs, mapping NVD records to their corresponding VFCs is challenging due to factors such as the following:

- **Limited Git links in NVD references:** Only a small percentage of NVD records include Git links that are marked with Patch tag, even if all these links are assumed to be true VFCs. For the majority of NVD records, alternative strategies are required, such as analyzing other reference links or seeking external security databases. One potential approach involves searching in relevant GitHub repositories.
- **Extensive commit history in OSS repositories:** Even with CPE (Common Platform Enumeration) information from NVD records, identifying relevant VFCs in OSS GitHub repositories is daunting. Large repositories often contain hundreds of thousands of commits, making it challenging to pinpoint the specific commits addressing vulnerabilities.
- **Semantic disparity between NVD descriptions and VFCs:** NVD descriptions have been employed to identify VFCs from a pool of commits [25]. However, these descriptions are often verbose, whereas commit messages are typically succinct. Additionally, while code changes in VFCs offer valuable context, aligning them with natural language descriptions poses challenges due to the differing modalities (code vs. text), complicating the matching process.

2.4 NVD Record Categories

To systematically identify VFCs, we categorize NVD records based on the types of links in their reference sections. Since VFCs are typically associated with Git, the presence or absence of Git links is a critical factor in categorizing NVD records. Additionally, as the NVD assigns a Patch tag to certain reference links, whether a reference includes this tag is another important criterion.

To facilitate our manual investigation, we define four mutually exclusive categories of NVD records based on these two factors:

- **Category 1:** NVD records containing at least one Git reference with a Patch tag: Commits associated with these Git references could be considered VFCs with high confidence.

- **Category 2:** NVD records containing at least one Git reference, but none of them have a Patch tag: While these non-patch-tagged Git references are likely to contain VFCs, the NVD has not yet confirmed their validity.
- **Category 3:** NVD records with no Git references but containing at least one other reference with a Patch tag: While these references are highly relevant to VFCs, it remains unclear how to extract VFCs from them.
- **Category 4:** NVD records with no Git references and no other references with a Patch tag: These are the remaining NVD records. These cases can be the most challenging among all the NVD records, as they lack any explicit relationship with VFCs and may be entirely irrelevant.

Considering these four categories, the challenges are twofold: for NVD records with Git references (Category 1 and 2), we need to manually verify whether the references correspond to true VFCs. Meanwhile, for NVD records with only non-Git references (Category 3 and 4), we need to first follow those links to uncover any potential Git references that could lead to VFCs; then, we also manually verify their precision.

3 Study Design

In this section, we first introduce the RQs; second, we present the data collection process; next, we provide the evaluation metrics adopted in each RQ; and finally, we provide the implementation details.

3.1 Research Questions

RQ1: What methods do humans use to identify VFCs associated with a given NVD record?

Motivation: Understanding how humans manually identify VFCs from NVD references is foundational for developing automated techniques. In this RQ, we present a study on manual VFC identification using only the existing reference links in NVD records. The goal of this investigation is to identify and categorize the different types of references found in NVD records and assess the potential of each reference type in uncovering VFCs. By manually analyzing the samples of NVD records and their associated references, we aim to gain insights into the characteristics and patterns of references that are most likely to lead to the identification of VFCs. This manual process will help us establish a set of rules and heuristics that can form the basis for automating the VFC identification process.

Approach: We sampled a representative set of NVD records from each category (as shown in Section 2.4) to maintain a 95% confidence level with a 5% margin of error. Subsequently, we manually identified VFCs within the sampled NVD records. Two authors independently identified VFCs from the NVD and meticulously documented their processes. For all categories of NVD records, they prioritized reviewing Git references and references tagged with Patch. If VFCs could not be located within these references, they proceeded to examine the remaining non-Git references and those without the Patch tag. Both authors are research engineers with over four years of programming experience, ensuring a high level of expertise in the identification process.

RQ2: What proportion of records in the NVD include patch links in their reference sections?

Motivation: RQ1 revealed that VFCs can be reliably identified from NVD references, especially Git references, regardless of Patch tags. To scale this process, we developed an automated pipeline to extract VFCs systematically, aiming to quantify their effectiveness and precision across all NVD record categories.

Approach: We directly extract commits from the Git references (Categories 1 and 2) as VFCs. There are also cases where Git references are issues or pull request links. For these cases, we extract all the commits inside these issues or pull request links by using GitHub, GitLab, or Bitbucket APIs and consider them as VFCs.

For NVD records without Git references (Category 3 and 4), we still explore the non-Git references by designing a VFC filtering heuristic that leverages domain-specific rules to refine the identification of true VFCs.

To evaluate the precision of these automatically extracted candidate VFCs, we conducted a manual analysis to verify their correctness. For this purpose, we sampled a representative subset of NVD records, ensuring a 95% confidence level with a 5% margin of error.

RQ3: How many patch links can be discovered from external security databases?

Motivation: While the NVD is a primary source of vulnerability information, it is not exhaustive in its coverage of VFCs. Many NVD records lack explicit VFC links or references, which limits their utility for developers and researchers seeking to understand or remediate similar vulnerabilities. However, other security databases and platforms often host complementary CVE information, including additional references, patches, or commit links that may not be present in the NVD. By exploring these external databases, we aim to uncover additional VFCs that are missing from the NVD records.

Approach: We build an automated pipeline that begins with extracting CVE identifiers from NVD records. We then search these CVE identifiers across six common external security databases, including Snyk [9], GitHub Advisory [4], Ubuntu Security [10], Nifi Apache Security [5], Django Security [3], and OSV Dev [7]. Snyk is a well-established vulnerability database that provides detailed information on vulnerabilities across multiple programming languages and frameworks. GitHub Advisory is a platform that aggregates security advisories from various sources, offering a centralized repository of vulnerability information. Ubuntu Security, Nifi Apache Security, and Django Security are project-specific security resources that provide valuable insights into vulnerabilities affecting widely used open-source software. OSV Dev is a community-driven vulnerability database that collects and curates vulnerability information from multiple sources.

To assess the quality of the VFCs collected from the external security databases, we further conduct a manual assessment on a subset of VFCs with a 95% confidence interval and 5% margin of error.

RQ4: How many patch links can be identified through GitHub repository analysis?

Motivation: Despite our efforts to extract VFCs from the NVD and external security databases, we observed that a significant proportion of NVD records still lack explicit VFC links. This gap highlights the need for alternative approaches to identify VFCs. GitHub, as one of the largest platforms for hosting open-source projects, is a natural candidate for this exploration. Many vulnerabilities are patched directly in GitHub repositories, and these patches are often documented in commits, pull requests, or issues. By leveraging GitHub repository analysis, we aim to identify additional VFCs that are not referenced in the NVD or external databases.

Approach: To answer this RQ, we revisit the effectiveness of two state-of-the-art VFC search techniques, Prospector [36] and PatchFinder [25], in identifying VFCs for NVD records in their relevant GitHub repositories. Developed concurrently, these methods have not been directly compared until now. To assess their precision in VFC extraction, we first apply both techniques to our curated After-March-2024 dataset. As Prospector outperforms PatchFinder on this dataset, we proceed to test Prospector further on the Implicit VFCs subset. Finally, we deploy Prospector across all implicit NVD records - those lacking patch-tagged Git references.

While alternative techniques like VCMatch [41], PatchScout [39], and VFCFinder [20] exist, we chose Prospector and PatchFinder for their alignment with our study's needs. PatchFinder has demonstrated superior performance over VCMatch and PatchScout in its evaluation [25]. Although VFCFinder shows promise, it is less suitable for NVD records, as its inference phase relies on inputs formatted for GitHub GHSA or the OSV Vulnerability Database.

RQ5: How do the VFCs extracted from NVD references, external security databases, and GitHub repositories compare in terms of precision, coverage, and overlap?

Motivation: By employing multiple strategies to identify VFCs - ranging from automated extraction from NVD reference (S1) and external databases (S2) to GitHub repositories (S3) - we have gathered a diverse set of VFCs. However, it is unclear how these sources compare in terms of their precision (the correctness of the VFCs identified from each source), coverage (the proportion of NVD records for which VFCs are found), and overlap (the extent to which VFCs from different sources are unique or redundant). Understanding these dimensions will also provide insights into the complementary nature of these sources and inform future efforts to automate and optimize VFC discovery.

Approach: In prior RQs, we have run different methods on NVD references, external security databases, and GitHub repositories, and we also conducted separate manual verification processes. In this RQ, we first compare the precision of each to draw comparisons. Next, we compare the coverage of VFCs extracted from these sources. We calculate the proportion of NVD records for which VFCs are found in each source and identify the unique contributions of each approach. Finally, we analyze the overlap between the VFCs extracted from the three sources to determine the extent to which they complement or duplicate each other. We use Venn diagrams to visualize and quantify the overlap, providing a clear picture of how these sources contribute to the overall set of VFCs. This comprehensive comparison will enable us to draw meaningful conclusions about the effectiveness of each approach and to identify opportunities for improving VFC discovery in the future.

3.2 Dataset Collection

To collect data for this study, we implemented a script to automate the retrieval of detailed information about NVD records using the NVD API up to March 28th, 2024. This tool streamlines the identification of VFCs by extracting key details from the NVD. Specifically, in our study, we mainly leverage the following pieces of information: (1) descriptions, (2) references, and (3) Common Platform Enumeration (CPE) configurations.

The descriptions offer critical insights into the nature of each vulnerability, including the potentially affected code segments and libraries. The references section includes links to technical records, vendor advisories, and, most importantly, patches or updates, which often contain or directly point to the relevant VFCs. Finally, the CPE configurations identify the specific affected products and versions.

3.3 Evaluation Metrics

In this section, we present the evaluation metrics we use to answer different RQs.

3.3.1 Manual Identification. In RQ1, to evaluate the effectiveness of manual annotators in identifying VFCs from NVD references, we define the success rate as follows:

$$\text{Success Rate} = \frac{\text{\#NVD Records with at least 1 identified VFC}}{\text{\#Sampled NVD Records}} \times 100\% \quad (1)$$

Here, the numerator represents the number of NVD records where at least one VFC was successfully identified by the annotator, while the denominator corresponds to the total number of NVD records in the sampled dataset. This metric provides a quantitative measure of how successfully annotators can identify VFCs within the given sample.

3.3.2 Automated Identification. In RQ2-4, we have run different automated methods to extract VFCs. To measure their effectiveness, we have performed manual verification on a sample of NVD records and the identified VFCs.

We report precision metrics from two perspectives:

- Precision_{Records}: Accuracy of mapping NVD records to at least one true VFC.
- Precision_{VFCs}: Accuracy of identified VFC candidates being true VFCs.

$$\text{Precision}_{\text{Records}} = \frac{\#\text{NVD Records with at least 1 confirmed true VFC}}{\#\text{Sampled NVD Records}} \times 100\% \quad (2)$$

$$\text{Precision}_{\text{VFCs}} = \frac{\#\text{Confirmed true VFCs}}{\#\text{Candidate VFCs}} \times 100\% \quad (3)$$

3.3.3 GitHub Repository Search Method Comparison. For RQ4, we use Recall@K (Formula 4) to compare the performance of PatchFinder and Prospector on the After-March-2024 dataset. Recall@K is also used by the authors of PatchFinder. It measures the proportion of true VFCs retrieved within the top-K predictions, with higher values indicating better performance.

$$\text{Recall@k} = \frac{\#\text{True VFCs in Top-k predictions}}{\#\text{All VFCs}} \quad (4)$$

3.3.4 Comparison Among Different Sources. For RQ5, we compute coverage to show how many NVD records can be successfully mapped to at least one VFC by combining all the sources.

$$\text{Coverage} = \frac{\#\text{NVD Records with at least 1 identified VFC}}{\#\text{All NVD Records}} \quad (5)$$

3.4 Implementation Details

3.4.1 RQ1: Manually finding VFCs. At the beginning of the VFC identification process for each NVD record, participants are instructed to read the NVD description and CWE information to gain a basic understanding of the vulnerability. Following this, they proceed to review the CPE sections to gather information about the affected projects. Participants then attempt to identify VFCs using all available resources, typically beginning with the references section of the NVD, which is the most probable source of VFCs. When examining the references, participants are advised to prioritize them in the following order: patch-tagged Git links, non-patch-tagged Git links, patch-tagged non-Git links, and non-patch-tagged non-Git links. If no VFCs are found within the NVD references, participants are encouraged to extend their searches to other online sources. However, to ensure efficiency, the search for each CVE is limited to a maximum of 10 minutes. At the end of each search, participants are required to complete an answer sheet, providing details such as the time spent on the search (in seconds), the links to any identified VFCs, and the methods used to find them.

3.4.2 RQ2: Finding VFCs from Reference. The heuristic begins with a web scraper to extract and analyze all references within each NVD record. It then identifies and retrieves the inner references associated with these records to construct the NVD references tree. This tree is built by recursively extending each reference through its own linked references. Notably, we start with the NVD-provided references as the first level and expand the tree by their references. For example, if an NVD reference points to a security advisory or bug tracker, the heuristic further explores the references within those linked resources to identify additional candidate VFCs. In other words, the reference tree is formed by recursively traversing the references and their linked references to expand its search space, increasing the likelihood of encountering a VFC. The depth parameter controls the extent of this recursive exploration, allowing for a balance between thoroughness and computational efficiency. At each level of the tree, the heuristic extracts relevant information from the references, such as Git commits or pull requests, which are then filtered based on predefined criteria to identify candidate VFCs. This iterative process continues until a specified depth is reached or a VFC is found. However, due to hardware and runtime constraints, our experiments limit the depth of the tree to two levels.

We apply a two-step filtering process once the references are extracted and organized. First, we retain high-confidence references, such as commit and issue links, from Git-based repository hosting services like GitHub, GitLab, and Bitbucket. Second, we cross-validate these references against the affected product information extracted from the CPE in the NVD record. If a Git link is determined to originate from the same repository as the CPE, it is classified as a candidate VFC. This validation step ensures that the identified commits are contextually relevant and directly associated with the reported vulnerability. This process of cross-validation is applied to all the NVD records.

3.4.3 RQ3: Finding VFCs from External Security Databases. For resources with APIs, such as the GitHub Advisory Database and OSV Dev, we query their APIs for each CVE to extract Git links from their reference sections. For external security databases without APIs, we deploy a tailored web scraper for each, applying the filter heuristics outlined in Section 3.4.2. These heuristics confirm that collected VFCs originate from Git-based hosting services and align with the CPE information of the targeted NVD records. As with RQ1, we identified all potential candidate VFCs to evaluate their coverage and performed a manual analysis to verify their accuracy, thus assessing precision.

3.4.4 RQ4: Finding VFCs from GitHub Repositories. We provide a brief description and the specific implementation we conducted on the two tools, i.e., PatchFinder [25] and Prospector [36] as follows.

PatchFinder [25]: (1) *Description:* PatchFinder has two phases: Phase 1 (initial retrieval) and Phase 2 (re-ranking). In Phase 1, PatchFinder uses a hybrid patch retriever that combines lexical matching via TF-IDF and semantic matching using a pre-trained CodeReviewer [26] model. This phase narrows down the candidate set by extracting commits that are similar to the CVE descriptions in both lexical and semantic aspects. Phase 2 involves re-ranking these candidates using a fine-tuned CodeReviewer model to learn the semantic correlations between CVE descriptions and commits, to improve the accuracy of the patch identification. (2) *Implementation:* We first restructure our After-March-2024 explicit dataset to have the same format as the authors' dataset. Specifically, for each NVD record found with VFCs, we follow PatchFinder's setting by randomly sampling 5,000 other non-patch commits from the same repositories to put into our dataset. We also construct another dataset containing NVD records before March 2024, to fine-tune Phase 2 of PatchFinder. The fine-tuning dataset includes all the explicit VFCs (patch-tagged Git references) associated with the 15,732 NVD records in Category 1. After the fine-tuning, we run the inferences of all two

phases of PatchFinder to get the output candidate VFCs, with the same settings mentioned in the PatchFinder paper.

Prospector [36]: (1) *Description:* Prospector, on the other hand, is a purely heuristic approach that is also developed to map vulnerability advisories, mainly NVD records, to the corresponding VFCs in public GitHub repositories. The methodology of Prospector involves retrieving vulnerability advisories, processing their descriptions to extract relevant keywords, file names, and method references, and then scanning source code repositories for commits that might correspond to these vulnerabilities. Using a set of heuristics, Prospector ranks candidate commits by evaluating the presence of certain indicators, such as the inclusion of vulnerability identifiers in commit messages or references to bug-tracking tickets and GitHub issues. This ranking is further refined by considering the changes made in the code itself.

(2) *Implementation:* We run the heuristic version of Prospector as described in their original paper [36]. Although Prospector has a large language model (LLM) support feature in their GitHub repository [8] to enhance the quality of the results, we decided not to use it for two reasons. First, this feature is a wrapper for calling APIs of commercial LLMs, which is not cost-effective to run on a large database such as NVD. Secondly, to date, the authors of Prospector have not published any results related to this LLM-support feature, as the evaluation from their paper is conducted on the only heuristic version of Prospector. After getting the candidate VFCs output from Prospector, we only keep VFCs with Prospector scores over 60 to ensure the quality and reduce manual effort to check their validity. We also modified Prospector heuristic rules to not prioritize commits found in GitHub issues and release links, as we have manually observed many commits in these cases are not VFCs. To run Prospector for each NVD record, we need to provide the CVE ID and its corresponding GitHub repository link as inputs.

Evaluation Datasets: It is important to note that existing works have primarily been evaluated on datasets of **known** VFCs, which were already identified as VFCs at the time of writing. However, the potential for data leakage remains unclear, particularly given that Prospector relies on heuristic rules; they might inadvertently evaluate data where those heuristic rules were initially formulated.

Our objective, in contrast, is to discover new and *hidden* VFCs - those that were unlabeled at the time of their publication. To this end, we focus on assessing performance specifically on such VFCs. To achieve this, we construct two datasets:

- **After-March-2024:** To evaluate the generalizability of PatchFinder and Prospector, we constructed a dataset comprising NVD records from Category 1—those featuring at least one Git reference tagged with Patch. This dataset includes records published after the release of these two tools in March 2024, extending through September 2024. We only include the NVD records belonging to Category 1, i.e., containing at least one patch-tagged Git reference; the rationale is that we achieve the highest precision in terms of NVD records and VFCs in both manual (RQ1) and automated identification (RQ2). This dataset aims to evaluate the tools automatically since we already have the ground-truth VFCs.
- **Implicit VFCs:** The second dataset comprises NVD records from Categories 2–4, which lack Git references tagged as Patch. Although these categories encompass 219,609 NVD records in total, Prospector requires two input parameters for each record: the CVE ID and the GitHub URL of the corresponding source repository [36]. Consequently, we include only those implicit NVD records for which public GitHub repository links are available, resulting in a dataset of 99,345 records, termed the *Implicit VFCs* dataset. The evaluation focuses exclusively on these implicit records from Categories 2, 3, and 4, as verifying the correctness of candidate VFCs for these records is inherently more complex than for the previous dataset, which benefits from established ground-truth VFCs. Since these NVD records lack explicit ties to ground-truth VFCs, we rely on a manual

verification process. We randomly selected 300 NVD records for which Prospector has generated VFC predictions and documented the outcomes of manual verification. Annotators review the candidate VFCs generated by the method under study to assess their validity.

4 Results

This section presents the results of our empirical study, addressing each RQ in sequence to evaluate the feasibility and challenges of mapping NVD records to their VFCs.

4.1 RQ1: Manual Investigation

Table 1. Statistics of NVD record categories and manual VFC extraction results.

Category	# Records	# Sampled	A#1	A#2
C1: Git Ref. with Patch tag	15,732	375	369 (98.4%)	369 (98.4%)
C2: Git Ref. without Patch tag	2,813	339	294 (86.7%)	302 (89.1%)
C3: No Git Ref. but has Patch tag	51,958	382	51 (13.3%)	37 (9.7%)
C4: No Git Ref. and no Patch tag	164,838	384	33 (8.6%)	24 (6.3%)

Note: The “Sampled” column represents the NVD records selected for manual identification. “A#1” and “A#2” indicate the number of NVD records where Annotator 1 and Annotator 2 successfully identified VFCs, respectively. Values in parentheses denote the success rate. “Ref.” refers to references.

Table 1 shows the results of manual VFC identification across the four NVD record categories defined in Section 2.4. In total, we have crawled 235,341 NVD records. The table also includes the results of the manual investigation.

4.1.1 NVD Records with Git References.

- Category 1: Git references with Patch tag.** Both annotators successfully identified VFCs in 98.4% of the 375 sampled records, reflecting high reliability. A small proportion of failed cases are the broken and inaccessible Git references, which are potentially due to repository deletion, renaming, privatization, or commit history changes (e.g., rebasing or squashing).
- Category 2: Git references without Patch tag.** Annotators achieved success rates of 86.7% (294/339) and 89.1% (302/339), with a Cohen’s kappa of 0.84 indicating strong inter-annotator agreement. Despite the absence of Patch tags, most Git references proved to be true VFCs upon verification. The few failed cases of this category are due to irrelevant Git references, such as links to issues, releases, or repository home pages, rather than actual VFCs. This may be explained by the fact that these Git references were submitted to the NVD, but the NVD has not yet analyzed them, resulting in the absence of Patch tags. This also suggests that future works can leverage these references as VFCs with high confidence.

Insight 1: In NVD records with Git references, the presence of the Patch tag is not critical for identifying VFCs. Commits from both tagged and untagged Git references serve as reliable candidate VFCs.

4.1.2 NVD Records without Git Reference.

- Category 3: No Git references but has Patch tag.** For NVD records with patch-tagged non-Git references, both annotators encountered greater difficulty, with success rates dropping to 13.3% and 9.7%, and a Cohen’s kappa of 0.77, indicating moderate agreement. This highlights the increased

effort required to identify VFCs in this category, even if they were confirmed to be highly relevant to VFCs.

- **Category 4: No Git references and no Patch tag.** Similarly, for NVD records with non-patch-tagged non-Git references, both annotators identified VFCs at even lower rates of 8.6% and 6.3%, with a Cohen's kappa of 0.79. Merged together, the annotators successfully identified VFCs in only 34 out of the 384 NVD records (8.85%), reflecting considerable variability in annotator agreement and the significant challenges of identifying VFCs without Git references or Patch tags.

The main reason for the lower number of VFCs found in Categories 3 and 4 compared to Categories 1 and 2 is that, without Git references, the annotators had to rely on more indirect and time-consuming methods to find VFCs. To be more specific, finding VFCs in Categories 3 and 4 required the annotators to go through different non-Git references to get every hint that could lead to VFCs. For example, there is a case where one of the annotators found the release tag of the fixed version for an NVD record. He then used this release tag to trace back the latest commit before the tag and manually double-checked again to ensure this commit was the true VFC. Another reason is that a significant portion of the affected products associated with these NVD records are not open-source, making it impossible to trace corresponding VFCs.

***Insight 2:** For NVD records without Git references, the Patch tag does not significantly aid VFC identification. The challenges in identifying VFCs from mining NVD records with non-Git references lie in the limited availability of publicly accessible version-control data, the complexity of cross-referencing multiple documents, and the reliance on extensive manual investigation. Nonetheless, the commits extracted from these records can still be considered potential VFCs.*

4.1.3 Potential Sources of VFCs. One of the main goals of the manual analysis is to identify sources of VFCs beyond patched Git references in the NVD. This approach helps address gaps and increases the number of VFCs that can be identified in the NVD, particularly in the absence of Git references. To achieve this, the annotators further extract information from the NVD records belonging to either Category 3 (Patch-tagged non-Git reference) or Category 4 (Non-Patch-tagged non-Git reference). The annotators then perform open card sorting to categorize the sources used to identify VFCs. At the end of this process, three categories were established: *By references*, *By Git*, and *By external resources*. In the following section, we will first describe each category with examples and then present their distributions.

By references refers to the process in which the annotators review all references from the NVD, such as URLs, advisories, or third-party records, to identify VFC links. The annotators may also follow additional links from these references and repeat the search process. However, due to the time constraint of 10 minutes per CVE, annotators are limited to searching a maximum of three inner links from the original references. An example of this process is illustrated in Figure 1: In the reference section of NVD for CVE-2011-2505, we examine all the reference URLs and identify a GitHub commit link from one of the non-Git references. Upon manual review, we confirm that this commit from the public repository of the affected product (phpMyAdmin) addresses the vulnerability described in the CVE (Swekey-related session manipulation). Therefore, we can confirm that this commit is the VFC for CVE-2011-2505.

By Git refers to the process where, for each CVE, the annotators extract key keywords from its NVD description and CWE description. The annotators then attempt to explore the public Git repository link for the affected product and search its Git commits for messages containing one or more of the extracted keywords. From this set of Git commits, the annotators conduct a final manual review to identify the VFCs. Figure 2 illustrates an example of manually identifying the VFC for CVE-2020-28364 through Git repositories. In this example, as there is no candidate VFC that



Fig. 1. Example of manually finding the VFC for CVE-2011-2505 by references

could be found in the references, we try to find if there are any accessible public Git repositories from the affected products of this CVE. Fortunately, through the CPE information of NVD, we can find public GitHub repositories of Locust, which is the affected product. Since the vulnerability of the CVE is cross-site scripting, we search for a set of keywords related to cross-site scripting in this repository and find a commit whose message contains “XSS”. After that, we again conduct a manual check to verify the correctness of this commit. To be more detailed, we see that the code diff of this commit fixing the XSS vulnerability in the “web.py” file particularly fits the CVE description, which describes an XSS vulnerability in the Web UI of Locust. In addition, the author date of this commit, which is October 22nd, 2020, is also very close to the CVE published date, which is November 9th, 2020. With these justifications, it can be confirmed that this commit is indeed the true VFC of CVE-2020-28364.

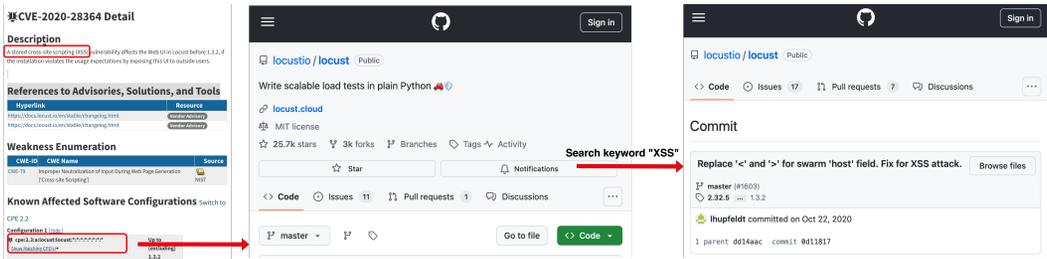


Fig. 2. Example of manually finding the VFC for CVE-2020-28364 by Git

By external resources refers to the process in which, for each CVE, the annotators extend their search beyond the NVD by exploring other resources, websites, and security advisories, such as Bugzilla [1] or the Open Source Vulnerability Database (OSV) [7], among others. These external resources may contain VFCs or crucial information leading to VFCs that are not covered by NVD.

An example of using this approach for CVE-2021-26559 is shown in Figure 3. In this example, we cannot find any candidate VFC from the two above approaches. Therefore, we shift to external advisories beyond NVD, starting with Snyk.io [9]. In the reference section of Snyk advisories for this CVE, we find a GitHub commit link. Through the manual check, it is confirmed that this commit is indeed the VFC for CVE-2021-26559.

Figure 4 shows the distribution of VFC sources used by both annotators to identify VFCs. The data shows that the majority of NVD records, 102 in total (71.8%), have VFCs identified exclusively through NVD references, highlighting the significant role NVD references play in locating VFCs. VFCs found solely in Git rank second, accounting for 29 NVD records (20.4%), indicating that while



Fig. 3. Example of manually finding the VFC for CVE-2020-28364 through Snyk.io, an external advisories

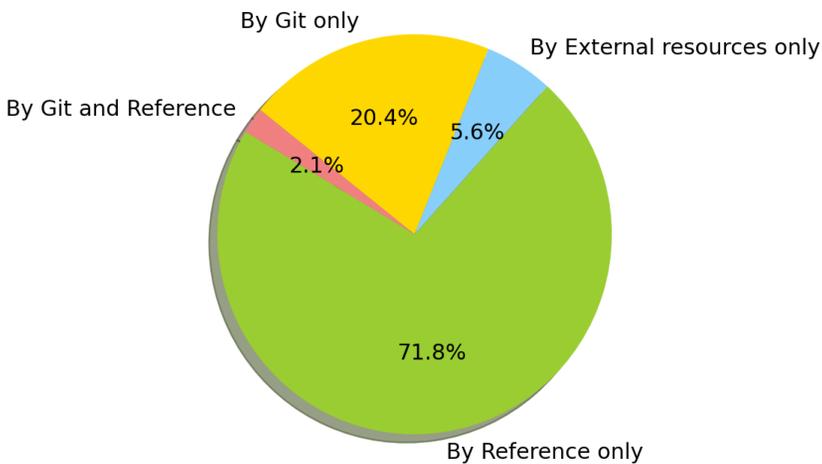


Fig. 4. Combined statistic on possible sources by both annotators

Git sources are essential, they are less frequently the sole resource for VFC discovery compared to NVD references. Furthermore, annotators find VFCs for only 8 NVD records (5.6%) exclusively through external resources. This suggests that while external resources may be useful for identifying VFCs, their sparsity makes it difficult for annotators to consistently identify the correct VFCs, resulting in their lower significance.

On the other hand, the statistics also reveal that a smaller subset of NVD records have VFCs identified from multiple sources. This refers to cases where one annotator identifies VFCs for an NVD record from one source. In contrast, the other annotator identifies VFCs for the same NVD record from a different source. Specifically, 3 NVD records (2.1%) have VFCs identified from both Git and NVD references. This suggests that identifying VFCs from multiple sources may increase the likelihood of discovering true VFCs.

Insight 3: Manual analysis identified three VFC source types: By references, By Git, and By external sources. NVD references dominated (71.8%), followed by Git (20.4%), with external resources contributing minimally (5.6%).

4.1.4 *Failure causes.* In this manual study, we also encountered several challenges that hindered the successful identification of VFCs for certain NVD records. The ease of finding NVD-VFC mappings varied significantly depending on the type and characteristics of the NVD records. For instance, the NVD records related to OSS hosted on popular version control platforms like GitHub were generally more straightforward to map to their VFCs, as the necessary information and references were more readily available. On the other hand, the NVD records associated with enterprise products or projects using non-Git version control systems proved to be more challenging, often requiring extensive manual effort and investigation. These challenges can be categorized into three main areas: repository accessibility, documentation and reference issues, and commit-related factors. Firstly, we were unable to find the repositories of affected products in some cases, particularly for enterprise products or those not using Git as their version control system. Secondly, the information provided in NVD descriptions and references was sometimes insufficient, too general, or redundant, making it difficult to locate the corresponding VFCs effectively. Lastly, commit-related issues posed significant obstacles, such as the possibility of VFCs being squashed or rebased, rendering them unavailable for analysis. Additionally, some commit messages were too vague or not directly related to the vulnerability. In contrast, in other cases, vulnerability fixes were part of multi-purpose commits, further complicating the identification process. These findings not only highlight the limitations of solely relying on NVD references for VFC identification but also provide valuable insights into the characteristics and challenges associated with the NVD-VFC mapping problem. By understanding the factors that influence the ease or difficulty of identifying VFCs for different types of NVD records, future research can develop more targeted and effective approaches to address these challenges.

Insight 4: The manual investigation faced challenges in identifying VFCs for certain NVD records, with varying difficulty based on project type and version control system. The main obstacles were repository accessibility, insufficient documentation and references, and commit-related issues

4.2 RQ2: Finding VFCs from NVD References

Table 2. Statistics of VFCs extracted from NVD references using our filter heuristics.

Category	# Webpages	# Records	# Identified Records	# Candidate VFCs
C1: Git Ref. with Patch tag	–	15,732	15,732	23,377
C2: Git Ref. without Patch tag	–	2,813	2,813	5,872
C3: No Git Ref. but has Patch tag	61,238	51,958	833	1,303
C4: No Git Ref. and no Patch tag	314,550	164,838	982	1,390
Total	375,788	235,341	20,360	31,942

Note: “# Webpages” and “# Records” denote the number of webpages and NVD records crawled. “# Identified Records” denotes the number of NVD records associated with candidate VFCs.

4.2.1 Automated Extraction Result. Table 2 presents the numerical statistics of VFCs extracted by our automated pipeline across various categories of NVD records. The proportion of NVD records with at least one extracted VFC was calculated relative to the total number of processed records. In total, the pipeline was applied to 235,341 NVD records.

For NVD records containing at least one Git reference (Categories 1 and 2), commits were extracted from those references. Specifically, for patch-tagged Git references (Category 1), the pipeline identified 23,377 candidate VFCs from 15,732 NVD records. For non-patch-tagged Git references (Category 2), it extracted 5,872 candidate VFCs from 2,813 NVD records. For NVD records containing only non-Git references (Categories 3 and 4), the pipeline successfully crawled 61,238 and 314,550 web pages corresponding to 51,958 and 164,838 NVD records, respectively. From these, it identified 1,303 and 1,390 candidate VFCs, covering 833 and 982 NVD records, respectively. In total, the automated pipeline extracted 31,942 VFC candidates from 20,360 NVD records.

Table 3. Manual verification result: VFCs extracted from references across different categories of NVD records.

Category	# Records	# True Records	# VFCs	# True VFCs
C1: Git Ref. with Patch tag	375	369 (98.4%)	402	396 (98.5%)
C2: Git Ref. without Patch tag	316	283 (89.6%)	361	303 (83.9%)
C3: No Git Ref. but has Patch tag	264	242 (91.7%)	351	284 (80.9%)
C4: No Git Ref. and no Patch tag	277	241 (87%)	301	253 (84.1%)
Total	857	766 (89.4%)	965	840 (87.0%)

Note: “# Records” indicate the number of sampled records; “# True Records” refers to the number of records manually confirmed to contain at least one true VFC, with the number in parentheses representing the $\text{Precision}_{\text{Records}}$. “# VFCs” represents the total number of candidate VFCs; “# True VFCs” represents the subset of those confirmed as true VFCs, with the number in parentheses representing the $\text{Precision}_{\text{VFCs}}$.

4.2.2 Manual Verification Result. Table 3 presents the manual evaluation results. In detail, Git references with Patch tags (Category 1) demonstrate the highest precision, with 98.4% of records containing at least one true VFC and 98.5% of candidate VFCs being verified as true. This underscores the reliability of Git references tagged with Patch for VFC identification. For Git references without Patch tags (Category 2), 83.9% of candidate VFCs are confirmed as true VFCs, resulting in 89.6% of records containing at least one true VFC. This indicates that while Git references without Patch tags are still highly useful, they require more manual effort for validation compared to their patch-tagged counterparts.

In the Non-Git category (Category 3), the results were also strong, with 284 true VFCs (80.9%) identified out of a total of 351 VFCs examined. This corresponds to 242 NVD records, representing 91.7% of the total sample NVD records. For the non-patch-tagged Non-Git category (Category 4), a manual review of 253 VFCs revealed that 301 VFCs (84.1%) were correctly identified as true VFCs, corresponding to 241 NVD records with at least one confirmed VFC (87% of the total sample NVD records).

In total, by merging the VFCs extracted from NVD references from all NVD record categories, we obtain a total of 31,942 VFCs across 20,360 NVD records, with an average precision of 90.0% and 87.9% for VFCs and NVD records, respectively. Prior studies usually only focus on Git references (Category 1 and 2), by adding extraction on non-Git references (Category 3 and 4), we increase the size by 9.2% in terms of VFCs and 9.8% in terms of NVD records. These results demonstrate the potential of further extraction of the non-Git reference in improving vulnerability datasets.

However, it is also worth noting that even combining all the NVD records that we found VFCs from NVD references, they still only account for 8.7% of all NVD records, highlighting the need for further exploration and improvements.

Answer to RQ2: The automated extraction of VFCs from NVD references identifies a total of 31,942 VFCs from 20,360 NVD records, achieving the precision of 89.4% and 87.0%, respectively. However, the dataset represents only 8.7% of all NVD records, underscoring the need for further research and refinement.

4.3 RQ3: Finding VFCs from External Security Databases

Table 4. Statistics of NVD records and candidate VFCs extracted from external databases.

External Database	# NVD Records	# Candidate VFCs
Snyk [9]	18,521	27,583
GitHub Advisory [4]	12,675	19,215
Ubuntu Security [10]	3,167	4,022
Nifi Apache Security [5]	68	68
Django Security [3]	142	301
OSV Dev [7]	697	855
Merged	18,985	29,254

4.3.1 Automated Extraction Result. Table 4 presents the data statistics of external databases in identifying additional candidate VFCs for NVD records. Among these sources, Snyk [9] and GitHub Advisory [4] contribute the most significant number of NVD records, with 18,521 and 12,675, respectively, resulting in 5,522 and 4,697 candidate VFCs. Other sources, such as Ubuntu Security [10], Django Security [3], and OSV Dev [7], provide smaller numbers of NVD records but still add valuable diversity to the dataset, offering additional candidate VFCs that the primary databases may not capture. To compare with other approaches, we merge the VFCs extracted from all the external sources together into a single dataset. When different external sources provide different VFCs for the same NVD record, the union of these VFCs is included in the merged dataset. Overall, the combined dataset for external security databases covers 18,985 unique NVD records, resulting in a total of 29,254 candidate VFCs, highlighting the value of leveraging diverse external security databases to enrich the dataset.

4.3.2 Manual Verification Result. Table 5 presents our manual assessment result. For the Snyk.io Database, over 442 VFCs are reviewed, 87.7% confirmed as true VFCs, associating to 344 NVD records (91.2% of the total sample NVD records). The GitHub Advisory Database shows similar precision, with 94.4% (352 of 373) of NVD records containing at least one true VFC and 92.5% (407 of 440) of individual VFCs verified as correct. Other sources, including Ubuntu, Apache NiFi, Django, and OSV Dev, also demonstrate high precision, with the percentage of true NVD records ranging from 86.0% to 94.4% and true VFCs from 86.4% to 92.5%. These results highlight the high quality of VFCs identified from external security databases, particularly the GitHub Advisory Database and Snyk.io, which serve as valuable complements to the NVD in identifying implicit VFCs for NVD records.

Table 5. Manual verification result: VFCs identified from external databases.

External Database	# Records	# True Records	# Candidate VFCs	# True VFCs
Snyk [9]	377	344 (91.2%)	442	388 (87.7%)
GitHub Advisory [4]	373	352 (94.4%)	440	407 (92.5%)
Ubuntu Security [10]	343	295 (86.0%)	401	344 (85.8%)
Nifi Apache Security [5]	58	52 (89.7%)	58	52 (89.7%)
Django Security [3]	104	91 (87.5%)	275	240 (87.2%)
OSV Dev [7]	248	220 (88.7%)	296	256 (86.4%)
Merged	1,472	1,334 (90.6%)	1,890	1,671 (88.4%)

Note: “# Records” indicate the number of sampled records; “# True Records” refers to the number of records manually confirmed to contain at least one true VFC, with the number in parentheses representing the $\text{Precision}_{\text{Records}}$. “# VFCs” represents the total number of candidate VFCs; “# True VFCs” represents the subset of those confirmed as true VFCs, with the number in parentheses representing the $\text{Precision}_{\text{VFCs}}$.

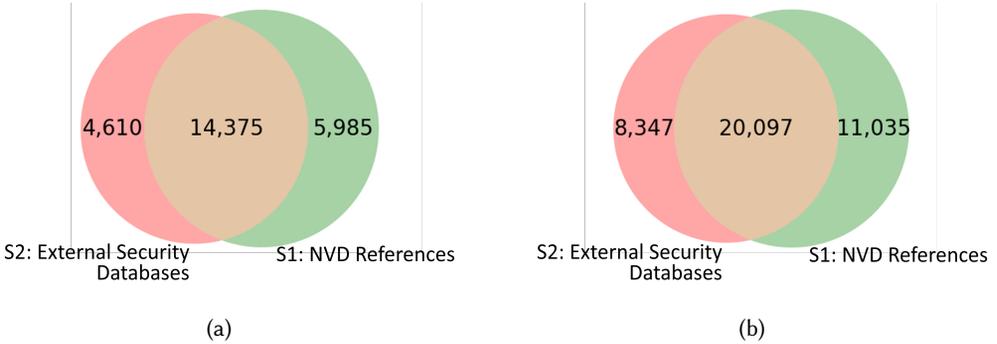


Fig. 5. Venn diagrams comparing NVD records (a) and VFCs (b) between NVD References and External Security Databases.

4.3.3 Comparing with NVD. Since the NVD and other external security databases exist independently, there may be questions about the overlap rate between the NVD records and VFCs across these different databases. We, therefore, further compared the VFCs extracted from the NVD references and those extracted from other security databases. We collectively refer to other security databases altogether as *External Security Databases*.

Figures 5a and 5b illustrate the overlap between VFCs identified through NVD references (green) and the combined set of VFCs extracted from all external security databases (red). In Figure 5a, 14,375 VFCs are shared between the two sources, while 4,610 and 5,985 VFCs are uniquely identified by external security databases and NVD references, respectively. Similarly, in Figure 5b, 20,907 VFCs are shared, with 8,347 and 11,035 uniquely identified by external security databases and NVD references.

Interestingly, this finding contrasts with our results from the manual analysis (Section 4.1), where the annotators were able to identify only 5.6% number of VFCs from this approach, which is much fewer than 71.8% number of VFCs identified through NVD references. We suspect that these differences arise from two factors.

First, in the manual approach, annotators typically begin by examining NVD references before consulting external security databases. As a result, when a VFC is found in both sources, the

annotators often attribute its discovery to NVD references, since it was identified there first. Second, while our automated approach is designed to replicate the manual search process, it is inherently limited in capturing the full complexity and flexibility of manual investigation within NVD references. The automated reference scraping cannot explore all the potential connections in NVD references as comprehensively as human annotators, leading to fewer VFCs being found through NVD references in the automated approach. This limitation raises a new challenge in improving our automated VFC identification to further utilize the potential of NVD references for this task.

Answer to RQ3: *The automated extraction of VFCs from external security databases identifies a total of 29,254 candidate VFCs for 18,985 NVD records, achieving an average Precision_{Records} of 90.6% and Precision_{VFCs} of 88.4%, respectively.*

4.4 RQ4: Finding VFCs from GitHub Repositories

Table 6. Recall@k results of Prospector and PatchFinder on After-March-2024 dataset.

Recall@k	PatchFinder	Prospector
K=1	22.12%	75.61%
K=2	25.42%	80.42%
K=3	33.79%	83.39%
K=4	39.2%	85.52%
K=5	45.61%	86.18%
K=6	48.29%	86.93%
K=7	50.91%	87.25%
K=8	52.14%	87.54%
K=9	54.36%	87.92%
K=10	55.72%	88.16%

4.4.1 On the After-March-2024 dataset. Table 6 shows the performance of Prospector and PatchFinder on the dataset. Across all K values from 1 to 10, Prospector consistently outperforms PatchFinder by a significant margin. For example, at K=1, Prospector achieves a Recall@1 of 75.61%, compared to PatchFinder's 22.12%, indicating that Prospector is far more effective at identifying the correct VFC in the top-ranked result. Similarly, at K=10, Prospector achieves a Recall@10 of 88.16%, compared to PatchFinder's 55.72%, maintaining its superior performance as K increases. The steady improvement in recall scores for both methods as K increases reflects the benefit of considering a broader range of candidate VFCs. These results demonstrate that Prospector is more robust and accurate in tracing VFCs than PatchFinder, which is due to the difference between VFCs' considerations from the two tools. Prospector prioritizes commits that clearly link to vulnerability fixes, especially those mentioning vulnerability IDs in messages or referenced in advisories. It also values commits with bug-tracking or GitHub issue IDs from advisories, as they provide direct, verifiable links to vulnerabilities. These commits are considered the most reliable for identifying VFCs. Less important are commits with keywords from NVD descriptions or changes to files/methods mentioned in NVD, which add supporting context but are not as directly linked. PatchFinder, on the other hand, uses deep learning without specific filters to exclude false candidates, which may weaken its ability to connect NVD records to their true VFCs accurately.

Furthermore, the observed decline in PatchFinder’s performance may be attributable to the dataset difference. The VFCs in both the fine-tuning dataset and the After-March-2024 dataset are limited to patch-tagged Git references from NVD records, diverging from the training and evaluation data utilized in PatchFinder’s original paper. Relying on a learning-based framework trained with a random sample of 5,000 non-patch commits as negative examples, PatchFinder’s performance may suffer from inconsistencies in training data. This limitation could hinder its adaptability across varied vulnerability contexts, exposing a fundamental challenge in refining such methods for this task.

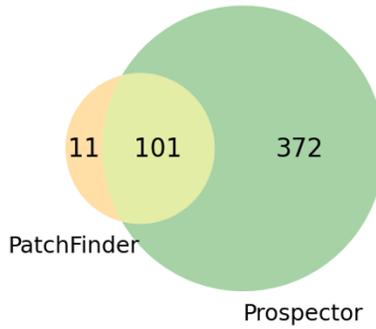


Fig. 6. Venn diagram on the results of PatchFinder and Prospector on the After-March-2024 dataset

The Venn diagram in Figure 6 presents another perspective of the comparison between PatchFinder and Prospector, illustrating the overlap between their Top-1 predictions. As shown, Prospector successfully identifies VFCs for 473 NVD records, significantly outperforming PatchFinder, which identifies VFCs for only 112 NVD records. Additionally, we observe a significant overlap of 101 NVD records between the two tools, indicating that Prospector also covers the majority of NVD records identified by PatchFinder. Notably, Prospector identifies 372 unique NVD records with true VFCs, whereas PatchFinder identifies only 11 unique NVD records, highlighting a substantial difference in coverage. This suggests that heuristic approaches, such as Prospector, are currently more reliable and robust for VFC identification from Git-hosting repositories, particularly when handling large-scale, real-world datasets.

4.4.2 *On the Implicit VFCs dataset.* We then evaluate Prospector on implicit VFCs to determine whether it can detect *more challenging* VFCs that have not yet been recorded in NVD. We did not evaluate the effectiveness of PatchFinder on the implicit VFC dataset due to the high computational cost of running PatchFinder and its limited effectiveness on the After-March-2024 dataset.

Table 7. Prospector results on both After-March-2024 explicit dataset and implicit dataset.

	# Records	# True Records	# VFCs	# True VFCs
After-March-2024	510	473 (92.75%)	597	473 (79.2%)
Implicit VFCs	300	220 (73.3%)	322	235 (73.0%)

Note: “# Records” and “# True Records” denote the number of NVD records found with VFCs by Prospector and true NVD records found with VFCs by Prospector. “# VFC” and “# True VFC” denote the number of candidate and true VFCs identified by Prospector.

Table 7 shows the results of Prospector. Prospector identifies 597 candidate VFCs with a precision of nearly 80% for explicit NVD records (on the After-March-2024 dataset). It also successfully associates these VFCs with 473 true NVD records, resulting in a precision of 92.75% in terms of NVD records. However, the precision drops on the Implicit VFCs dataset, with 73.0% for VFCs precision and 73.3% for NVD records precision. These results suggest that implicit VFCs are more challenging to identify than explicit VFCs when using Prospector.

Table 8. Results of Prospector running on a larger implicit dataset.

#NVD records processed	#NVD records found with VFCs	#VFCs
99,345	2,795	3,686

4.4.3 *Automated Extraction Result.* Given the precision of Prospector on our Implicit VFCs dataset is over 70%, we decide to run this method on the whole NVD records that do not have an explicit VFC mentioned in the reference, i.e., do not have a patch-tagged Git reference. Out of the total 99,345 NVD records with public GitHub repositories links processed, only 2,795 records are found to have associated VFCs that have a Prospector score over 60. This resulted in a total of 3,686 high-confidence VFCs.

Answer to RQ4: On our After-March-2024 dataset, Prospector significantly outperforms PatchFinder in identifying unseen explicit VFCs from GitHub repositories. On our Implicit VFCs dataset, Prospector experiences a precision drop, from 79.2% and 92.7% on the After-March-2024 dataset to around 73% for both VFCs and NVD records, respectively. These findings highlight that implicit VFCs are significantly more challenging to identify than explicit ones.

4.5 RQ5: Comparison Among Different Sources

Table 9. Statistics of our dataset by combining all the Steps.

Dataset	# Records	# VFCs	Precision _{Records}	Precision _{VFCs}
S1: From NVD Reference	20,360	31,942	89.4%	87.0%
S2: From External Databases	18,985	29,254	90.6%	88.4%
S3: From GitHub Repositories	2,795	3,686	73.3%	73.0%
Full Dataset	26,710	37,441	87.8%	86.1%

To address RQ5, we compare the effectiveness of the three primary sources used in this study - NVD references (S1), external security databases (S2), and GitHub repositories (S3) - in terms of precision, coverage, and overlap. This analysis synthesizes the results from RQ2, RQ3, and RQ4 to provide a holistic view of how these methods contribute to mapping NVD records to their VFCs.

4.5.1 *Precision.* We evaluate *precision* from two perspectives: Precision_{Records} (the percentage of sampled NVD records mapped to at least one true VFC) and Precision_{VFCs} (the percentage of identified candidate VFCs confirmed as true VFCs). Table 9 summarizes these metrics for each source and the combined dataset.

- S1: NVD References: This source yields a $\text{Precision}_{\text{Records}}$ of 89.4% and a $\text{Precision}_{\text{VFCs}}$ of 87.0% across 20,360 records and 31,942 VFCs. The high precision reflects the reliability of Git references, particularly those tagged with Patch, which provide direct links to VFCs. However, the automated pipeline's limitation to a two-level reference tree may exclude some valid VFCs from non-Git references, slightly hindering its precision.
- S2: External Databases: Extracting VFCs from six external security databases (e.g., Snyk, GitHub Advisory) results in a $\text{Precision}_{\text{Records}}$ of 90.6% and a $\text{Precision}_{\text{VFCs}}$ of 88.4% for 18,985 records and 29,254 VFCs. This source achieves the highest precision, likely due to the curated nature of these databases, which often include detailed patch information not present in the NVD.
- S3: GitHub Repositories: Using Prospector on the implicit VFC dataset, this source achieves a $\text{Precision}_{\text{Records}}$ of 73.3% and a $\text{Precision}_{\text{VFCs}}$ of 73.0% across 2,795 records and 3,686 VFCs. The lower precision compared to S1 and S2 highlights the challenge of identifying implicit VFCs, where heuristic-based methods struggle with less explicit links to vulnerabilities.

The full dataset, merging all three sources, achieves an overall $\text{Precision}_{\text{Records}}$ of 87.8% and $\text{Precision}_{\text{VFCs}}$ of 86.1%. This weighted average reflects the dominant contribution of S1 and S2, tempered by the lower precision of S3 due to its smaller scale and focus on harder-to-detect implicit VFCs.

4.5.2 Coverage. We define *coverage* as the proportion of NVD records successfully mapped to at least one VFC relative to the total NVD dataset. Table 9 shows the individual contributions:

- S1: NVD References: Maps 20,360 unique NVD records, covering approximately 8.7% of the NVD. This is the largest single-source contribution, driven by the automated pipeline's ability to extract VFCs from Git references and limited non-Git references.
- S2: External Databases: Identifies VFCs for 18,985 records, covering about 8.1% of the NVD. While slightly lower than S1, this source complements NVD references by capturing VFCs absent from the NVD, especially from well-maintained databases like Snyk and GitHub Advisory.
- S3: GitHub Repositories: Contributes only 2,795 records (1.2% of the NVD), reflecting the computational constraints and focus on implicit VFCs not found in S1 or S2. Despite its limited scale, S3 adds unique mappings that enhance overall coverage.

Combining all sources, the full dataset maps 26,710 unique NVD records to 37,441 VFCs from 7,634 projects, achieving a total coverage of approximately 11.3% of the NVD. This combined effort exceeds the 8.7% coverage from NVD references alone, demonstrating the value of integrating multiple sources. However, with over 85% of NVD records still unmapped, substantial gaps remain, particularly for records lacking Git references (Categories 3 and 4).

4.5.3 Overlap. To assess redundancy and uniqueness, we analyze the overlap of NVD records and VFCs across the three sources. In Section 4.3.3, we already compare the VFCs extracted from NVD references (S1) and external databases (S2), revealing significant overlap (e.g., 14,375 shared NVD records and 20,907 shared VFCs as shown in Figure 5). Here, we extend this analysis to include S3, focusing on pairwise comparisons - S1 versus S3 and the combined S1+S2 versus S3 - to assess how these sources complement one another in the merged dataset.

- S1 vs. S3: Figure 7a shows 810 NVD records common between NVD references and GitHub repositories, with 4,344 unique to S1 and 1,985 unique to S3. This moderate overlap suggests that the GitHub repository search complements the NVD references, and each source contributes unique VFCs, enhancing overall diversity.
- S1+S2 vs. S3: Figure 7b shows that 1,234 NVD records shared by S1+S2 (NVD reference and external security databases) and S3. This indicates that even by combining NVD references and

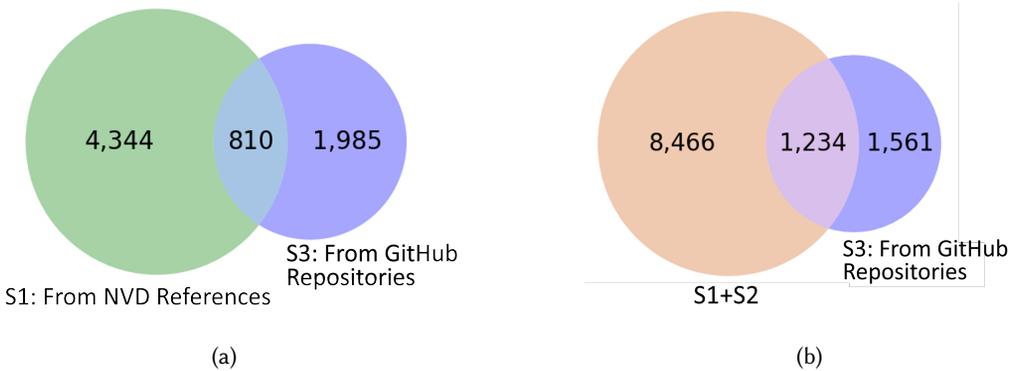


Fig. 7. Venn diagrams comparing at least one VFC found in NVD records by S1 vs. S3 (a) and S1+S2 vs. S3 (b)

external security databases, the GitHub repository search can still provide VFCs for unique NVD records.

The merged dataset of 26,710 records indicates that combining S1 and S2 captures the majority of mappings, with S3 providing a smaller but critical supplement. The overlap between S1 and S2 reduces the net gain from external databases, while S3’s unique additions highlight the potential of repository-based methods for hard-to-find VFCs.

Answer to RQ5: By leveraging NVD references, external databases, and GitHub repositories, we mapped 26,710 NVD records to 37,441 VFCs, achieving an overall $Precision_{Records}$ of 87.8% and $Precision_{VFCs}$ of 86.1%. External databases (S2) provide the highest precision (90.6% for records), followed closely by NVD references (89.4%), while GitHub repositories (73.3%) lag due to the complexity of implicit VFCs. Coverage reaches 11.3% of the NVD, with NVD references contributing the largest share (8.7%), supplemented by external databases (8.1%) and GitHub repositories (1.2%). Overlap between NVD references and external databases is significant, but GitHub repositories add unique mappings, enhancing the dataset’s comprehensiveness.

5 Discussion

5.1 Lesson Learned

Lesson #1: VFCs from Git references are not enough for constructing a vulnerability dataset. Our study reveals that VFCs identified through Git references come from 18,545 NVD records, which constitute only 7.9% of the total NVD records in the NVD dataset. This highlights the inadequacy of relying solely on VFCs from Git references for constructing vulnerability datasets, resulting in a small-scale dataset. Such limited datasets may introduce bias in vulnerability research and lead to poor performance in learning-based models.

Lesson #2: VFCs from NVD records without Git References should be mined to expand vulnerability datasets. In this study, we explore various methods to identify VFCs through manual investigation of related resources, such as NVD references, external databases, and GitHub repositories. Step 1 in our automated pipeline successfully identifies 29,249 VFCs from 18,545 NVD records with Git references. However, for NVD records without Git references, we were only able to identify 2,693 VFCs from 1,815 NVD records. It is noteworthy that although NVD records without Git references constitute the majority in the NVD database, the VFCs we identified from these records represent only around 0.77% of all NVD records, highlighting a substantial gap and room for

improvement. We recommend that future research prioritize mining VFCs from non-Git references to significantly enhance the number of vulnerable samples derived from the NVD. Broadening the scope beyond Git references will not only enrich the comprehensiveness of vulnerability datasets but also improve the effectiveness and robustness of vulnerability detection and analysis tools that depend on NVD data.

Lesson #3: Learning-based methods for VFC identification from NVD are not ready yet

Considerable effort has been invested in developing learning-based methods, utilizing either deep learning or LLMs, to detect VFCs within GitHub repositories. However, our evaluation reveals that PatchFinder, a leading state-of-the-art tool, delivers suboptimal performance on our dataset. Even after fine-tuning, PatchFinder achieves a Recall@1 of 22.12%. Furthermore, it is outperformed by Prospector, a purely heuristic-based approach, on the same dataset.

One plausible explanation for this disparity lies in the differences between the datasets. First, both the training and evaluation data utilized in our study differ from the ones used by PatchFinder. In addition, PatchFinder employs a random sampling of 5,000 non-patch commits as negative samples during its training. In our study, we fine-tuned PatchFinder using our explicit data before March, 2024, and similarly sampled 5,000 non-patch commits. Variations in the training data, combined with the non-heuristic nature of deep learning, may contribute to the observed inconsistencies in performance. These results underscore the necessity for further research and enhancement of learning-based techniques to advance the accuracy and reliability of VFC identification.

Lesson #4: Heuristic-based methods still face challenges in handling implicit VFCs. While Prospector significantly outperforms PatchFinder in identifying explicit VFCs on our After-March-2024 dataset, our experimental results indicate a notable drop in precision for the Implicit VFCs dataset. Specifically, it achieves only around 73% precision on both NVD records and VFCs. Although this precision is encouraging, we found it insufficient for reliably identifying implicit VFCs. Moreover, the significant execution times of several minutes per CVE also present practical challenges when handling over 200k NVD records in the NVD. In addition, we are only able to find VFCs for 2,795 NVD records, which is 1.2% of the total NVD records. This indicates that the prospector has a low recall rate when identifying VFCs.

5.2 Threats to Validity

5.2.1 Threats to Internal Validity. The manual investigation and verification in RQ1-RQ4 may introduce biases, as annotators might overlook certain VFCs due to methodological limitations or variability in interpretation. Additionally, ensuring the correctness of all identified VFCs is challenging, as it requires expertise across diverse vulnerability contexts. To mitigate these threats, annotators reviewed a large number of vulnerabilities to gain domain knowledge before labeling.

5.2.2 Threats to External Validity. The evaluation of automated tools like Prospector and PatchFinder was limited to the After-March-2024 dataset due to computational constraints. For example, preparing PatchFinder's dataset required approximately 10GB of storage and 28 hours of GPU time, making it impractical to scale to the entire NVD. Our results may not be generalized to other datasets. To mitigate this, we included all the explicit VFCs identified after March 2024 until September 2024. This evaluation dataset reflects varying repository structures, programming languages, and vulnerability types, ensuring the evaluation remains broadly representative.

5.2.3 Threats to Construct Validity. The selection of evaluation metrics could potentially introduce threats to construct validity in our study. To address this, we have employed a varied set of metrics tailored to different RQs. To assess whether a method accurately identifies VFCs, we evaluate precision from two distinct perspectives: the NVD record level and the VFC level. This dual approach

offers a more holistic view of each method's performance. Given that a single NVD record may correspond to multiple VFCs, calculating recall proves challenging and non-trivial. Consequently, we consider precision a suitable metric, effectively reflecting the performance capabilities of each method.

6 Related Work

In addition to the studies discussed in Section 2.2 that focus on mapping NVD/CVE records to their VFCs, other research has centered on identifying VFCs [29, 30, 39, 41, 42, 45]. These methods can be broadly categorized into two classes: VFC identification and silent VFC identification [38], depending on whether commit messages are explicitly utilized.

For VFC identification, tools like VulCurator [29] and Hermes [31] leverage various information sources, such as commit messages and GitHub issues, to classify VFCs. Recently, Bui et al. [16] proposed a large-scale VFC dataset from Java OSS. They manually annotated 784 VFCs and developed a set of keywords for filtering VFCs based on commit messages. Using this automated keyword-based filtering, their dataset expanded to include 16,837 VFCs.

Silent VFCs refer to VFCs without explicit log messages indicating the fixed vulnerability. Recent studies have introduced methods to identify these silent VFCs [30, 43, 44]. For instance, VulFixMiner [44] is the first deep-learning-based approach for detecting silent VFCs. It fine-tunes CodeBERT on cross-project and cross-language data to generate contextual embedding vectors for each commit based on code changes in affected files. These embeddings are then used to compute a prediction score, representing the likelihood that a commit addresses a vulnerability.

While these approaches advance VFC detection, they prioritize identification over tracing VFCs back to NVD records. Consequently, their applicability to our objective - mapping NVD records to their corresponding VFCs - is limited. Our work complements these identification-focused studies by addressing this mapping task, offering a broader perspective on VFC research.

7 Conclusion and Future Work

In this work, we conducted a comprehensive exploratory study to examine how hard to map NVD records to their corresponding VFCs. Our empirical study confirms that mapping NVD records to their VFCs is remarkably difficult. Across the sampled NVD records, our manual analysis achieved over 86% success with Git references but only 13% or less with non-Git ones. Our automated pipeline extracted 31,942 VFCs from 20,360 records (87% precision), external databases added 29,254 VFCs for 18,985 records (88.4% precision), and Prospector on GitHub mapped 3,686 VFCs for 2,795 records (73% precision). Despite these efforts, we linked just 26,710 records - 11.3% of the whole NVD records - leaving 88.7% unmapped, especially records with only non-Git references.

The core challenge lies in the NVD's reliance on sparse, explicit links: Git references cover only 7.9%, and external sources and repository searches struggle with precision and scale for the rest. Yet, our 37,441 VFCs provide a valuable starting point for analysis and detection.

Looking ahead, we plan to further enhance automation to parse non-Git references (e.g., via deeper scraping or NLP) and enhance the precision of the GitHub repository analysis tool to improve VFC identification and dataset quality. Additionally, we aim to leverage this dataset to advance other areas of vulnerability-related research, such as automated vulnerability detection and repair.

The replication package is available at <https://github.com/hungkien05/VFC-from-NVD-study>.

References

- [1] [n. d.]. Bugzilla Redhat. <https://bugzilla.redhat.com/>. (Accessed on 12/15/2024).
- [2] [n. d.]. CVE - Home. <https://cve.mitre.org/>. (Accessed on 12/15/2024).
- [3] [n. d.]. Django Security Reports. <https://docs.djangoproject.com/en/5.1/topics/security/>. (Accessed on 12/15/2024).
- [4] [n. d.]. GitHub Advisory Database. <https://github.com/advisories>. (Accessed on 12/15/2024).
- [5] [n. d.]. Nifi Apache Security. <https://nifi.apache.org/documentation/security/>. (Accessed on 12/15/2024).
- [6] [n. d.]. NVD - Home. <https://nvd.nist.gov/>. (Accessed on 12/15/2024).
- [7] [n. d.]. OSV database. <https://osv.dev/>. (Accessed on 12/15/2024).
- [8] [n. d.]. Prospector GitHub Repository. <https://github.com/SAP/project-kb/tree/main/prospector>. (Accessed on 12/15/2024).
- [9] [n. d.]. Synk.io. <https://synk.io/>. (Accessed on 12/15/2024).
- [10] [n. d.]. Ubuntu Security Reports. <https://https://ubuntu.com/security/>. (Accessed on 12/15/2024).
- [11] 2018. Flaw leads to Google+ shutting down. *Network Security* 2018, 10 (2018), 3. doi:10.1016/S1353-4858(18)30095-3
- [12] Jafar Akhoundali, Sajjad Rahim Nouri, Kristian Rietveld, and Olga Gadyatskaya. 2024. MoreFixes: A large-scale dataset of CVE fix commits mined through enhanced repository discovery. In *Proceedings of the 20th International Conference on Predictive Models and Data Analytics in Software Engineering*. 42–51.
- [13] Lingfeng Bao, Xin Xia, Ahmed E Hassan, and Xiaohu Yang. 2022. V-SZZ: automatic identification of version ranges affected by CVE vulnerabilities. In *Proceedings of the 44th international conference on software engineering*. 2352–2364.
- [14] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. 30–39.
- [15] S Biswas, M Sohel, MM Sajal, T Afrin, T Bhuiyan, and MM Hassan. 2018. A study on remote code execution vulnerability in web applications. In *International conference on cyber security and computer science (ICONCS 2018)*. 50–57.
- [16] Tan Bui, Yan Naing Tun, Yiran Cheng, Ivana Clairine Irsan, Ting Zhang, and Hong Jin Kang. 2024. JavaVFC: Java Vulnerability Fixing Commits from Open-source Software. *arXiv preprint arXiv:2409.05576* (2024).
- [17] Yizheng Chen, Zhoujie Ding, Lamyia Alowain, Xinyun Chen, and David Wagner. 2023. Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 654–668.
- [18] Long Cheng, Fang Liu, and Danfeng Yao. 2017. Enterprise data breach: causes, challenges, prevention, and future directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 5 (2017), e1211.
- [19] Yiran Cheng, Lwin Khin Shar, Ting Zhang, Shouguo Yang, Chaopeng Dong, David Lo, Shichao Lv, Zhiqiang Shi, and Limin Sun. 2024. LLM-Enhanced Static Analysis for Precise Identification of Vulnerable OSS Versions. *arXiv preprint arXiv:2408.07321* (2024).
- [20] Trevor Dunlap, Elizabeth Lin, William Enck, and Bradley Reaves. 2024. VFCFinder: Pairing Security Advisories and Patches. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. 1128–1142.
- [21] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. A C/C++ code vulnerability dataset with code changes and CVE summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 508–512.
- [22] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. 2022. VulRepair: a T5-based automated software vulnerability repair. In *Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering*. 935–947.
- [23] Daan Hommersom, Antonino Sabetta, Bonaventura Coppola, Dario Di Nucci, and Damian A Tamburri. 2024. Automated mapping of vulnerability advisories onto their fix commits in open source repositories. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 1–28.
- [24] Triet Huynh Minh Le, David Hin, Roland Croft, and M Ali Babar. 2021. Deepcva: Automated commit-level vulnerability assessment with deep multi-task learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 717–729.
- [25] Kaixuan Li, Jian Zhang, Sen Chen, Han Liu, Yang Liu, and Yixiang Chen. 2024. PatchFinder: A Two-Phase Approach to Security Patch Tracing for Disclosed Vulnerabilities in Open-Source Software. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 590–602.
- [26] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Singapore, Singapore) (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1035–1047. doi:10.1145/3540250.3549081
- [27] Ruyan Lin, Yulong Fu, Wei Yi, Jincheng Yang, Jin Cao, Zhiqiang Dong, Fei Xie, and Hui Li. 2024. Vulnerabilities and Security Patches Detection in OSS: A Survey. *Comput. Surveys* 57, 1 (2024), 1–37.

- [28] Peter Mell, Karen Scarfone, Sasha Romanosky, et al. 2007. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-forum of incident response and security teams*, Vol. 1. 23.
- [29] Truong Giang Nguyen, Thanh Le-Cong, Hong Jin Kang, Xuan-Bach D Le, and David Lo. 2022. Vulcurator: a vulnerability-fixing commit detector. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1726–1730.
- [30] Truong Giang Nguyen, Thanh Le-Cong, Hong Jin Kang, Ratnadira Widayarsi, Chengran Yang, Zhipeng Zhao, Bowen Xu, Jiayuan Zhou, Xin Xia, Ahmed E Hassan, et al. 2023. Multi-granularity detector for vulnerability fixes. *IEEE Transactions on Software Engineering* 49, 8 (2023), 4035–4057.
- [31] Giang Nguyen-Truong, Hong Jin Kang, David Lo, Abhishek Sharma, Andrew E Santosa, Asankhaya Sharma, and Ming Yi Ang. 2022. Hermes: Using commit-issue linking to detect vulnerability-fixing commits. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 51–62.
- [32] Georgios Nikitopoulos, Konstantina Dritsa, Panos Louridas, and Dimitris Mitropoulos. 2021. CrossVul: a cross-language vulnerability dataset with commit data. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1565–1569.
- [33] Mike O’Leary and Mike O’Leary. 2019. Privilege Escalation in Linux. *Cyber Operations: Building, Defending, and Attacking Modern Computer Networks* (2019), 419–453.
- [34] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. 2023. Fine-grained commit-level vulnerability type prediction by CWE tree structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 957–969.
- [35] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. 2015. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 426–437.
- [36] Antonino Sabetta, Serena Elisa Ponta, Rocio Cabrera Lozoya, Michele Bezzi, Tommaso Sacchetti, Matteo Greco, Gergő Balogh, Péter Hegedűs, Rudolf Ferenc, Ranindya Paramitha, Ivan Pashchenko, Aurora Papotti, Ákos Milánkovich, and Fabio Massacci. 2024. Known Vulnerabilities of Open Source Projects: Where Are the Fixes? *IEEE Security & Privacy* 22, 2 (2024), 49–59. doi:10.1109/MSEC.2023.3343836
- [37] Riccardo Scandariato, James Walden, Aram Hovsepian, and Wouter Joosen. 2014. Predicting Vulnerable Software Components via Text Mining. *IEEE Transactions on Software Engineering* 40 (10 2014), 993–1006. doi:10.1109/TSE.2014.2340398
- [38] Jiamou Sun, Zhenchang Xing, Qinghua Lu, Xiwei Xu, Liming Zhu, Thong Hoang, and Dehai Zhao. 2023. Silent vulnerable dependency alert prediction with vulnerability key aspect explanation. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 970–982.
- [39] Xin Tan, Yuan Zhang, Chenyuan Mi, Jiajun Cao, Kun Sun, Yifan Lin, and Min Yang. 2021. Locating the Security Patches for Disclosed OSS Vulnerabilities with Vulnerability-Commit Correlation Ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, Republic of Korea) (CCS ’21)*. Association for Computing Machinery, New York, NY, USA, 3282–3299. doi:10.1145/3460120.3484593
- [40] Dean Turner, Marc Fossi, Eric Johnson, Trevor Mack, Joseph Blackbird, Stephen Entwisle, Mo King Low, David McKinney, and Candid Wueest. 2008. *Symantec Internet Security Threat Report: Trends for July-December 07-Volume XIII*. Technical Report. Technical report, Symantec Corporation.
- [41] Shichao Wang, Yun Zhang, Liangfeng Bao, Xin Xia, and Minghui Wu. 2022. Vcmatch: a ranking-based approach for automatic security patches localization for OSS vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 589–600.
- [42] Congying Xu, Bihuan Chen, Chenhao Lu, Kaifeng Huang, Xin Peng, and Yang Liu. 2022. Tracking patches for open source software vulnerabilities. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 860–871.
- [43] Jiayuan Zhou, Michael Pacheco, Jinfu Chen, Xing Hu, Xin Xia, David Lo, and Ahmed E Hassan. 2023. Colefunda: Explainable silent vulnerability fix identification. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2565–2577.
- [44] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E Hassan. 2021. Finding a needle in a haystack: Automated mining of silent vulnerability fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 705–716.
- [45] Yaqin Zhou and Asankhaya Sharma. 2017. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 914–919.