VFCFinder: Pairing Security Advisories and Patches

Trevor Dunlap
North Carolina State University
Raleigh, North Carolina, USA
tdunlap@ncsu.edu

Elizabeth Lin
North Carolina State University
Raleigh, North Carolina, USA
etlin@ncsu.edu

William Enck
North Carolina State University
Raleigh, North Carolina, USA
whenck@ncsu.edu

Bradley Reaves
North Carolina State University
Raleigh, North Carolina, USA
bgreaves@ncsu.edu

ABSTRACT

Security advisories are the primary channel of communication for discovered vulnerabilities in open-source software, but they often lack crucial information. Specifically, 63% of vulnerability database reports are missing their patch links, also referred to as vulnerability fixing commits (VFCs). This paper introduces VFCFinder, a tool that generates the top-five ranked set of VFCs for a given security advisory using Natural Language Programming Language (NL-PL) models. VFCFinder achieves a 96.6% recall for finding the correct VFC within the Top-5 commits, and an 80.0% recall for the Top-1 ranked commit. VFCFinder generalizes to nine different programming languages and outperforms state-of-the-art approaches by 36 percentage points in terms of Top-1 recall. As a practical contribution, we used VFCFinder to backfill over 300 missing VFCs in the GitHub Security Advisory (GHSA) database. All of the VFCs were accepted and merged into the GHSA database. In addition to demonstrating a practical pairing of security advisories to VFCs, our general open-source implementation will allow vulnerability database maintainers to drastically improve data quality, supporting efforts to secure the software supply chain.

CCS CONCEPTS

• **Security and privacy** → Software security engineering.

KEYWORDS

Vulnerability Fixing Commit, Security Patches, Vulnerability Data

ACM Reference Format:
Trevor Dunlap, Elizabeth Lin, William Enck, and Bradley Reaves. 2024. VFCFinder: Pairing Security Advisories and Patches. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3634737.3657007

1 INTRODUCTION

Security advisories help users identify vulnerabilities, apply necessary fixes, and facilitate informed decision-making regarding components in software. The United States and the European Union have emphasized the need for high-quality advisories to address software dependency vulnerabilities effectively [34, 59]. Nevertheless, many existing security advisories lack crucial information [11].

Vulnerability fixing commits (VFCs) are a valuable but often missing part of security advisories. VFCs help practitioners mitigate vulnerabilities by enhancing software composition analysis tools [39, 40] and enabling patch presence verification [50, 53, 60], as well as new state-of-the-art techniques such as enabling few-shot bug repair [26, 57]. While the security community frequently focuses on identifying new vulnerabilities in code, less attention is given to identifying fixes for vulnerabilities [21, 51, 54]. This disparity is also reflected in practice. GitHub and Sonatype use human curators to enhance vulnerability databases [17, 48]; however, the volume of security advisories exceeds the available workforce, leading to 63% of advisories without patch links; see Figure 1.

Prior work established several variations for matching security advisories to VFCs. Initial approaches include extracting the vulnerability ID from commit messages [23] or following reference links in advisories [24, 58]. However, poorly documented security commit messages [43] and incomplete security advisories [11] limit the effectiveness of these techniques. In response to these limitations, machine learning approaches have shown promise by transitioning the task into a ranking problem. For instance, FixFinder [21] ranks commits using 23 features and a logistic regression model, achieving a Top-1 recall of 65.1% and Top-5 recall of 77.7% on a single Java dataset [41]. PatchScout [51] uses 22 features and RankNet [5] to attain a Top-1 recall of 69.5% and Top-5 recall of 85.4% across various C/C++ projects and a single Java project. VCMatch [54], and its GUI-based implementation Patchmatch [47], extends PatchScout using 100 features and three machine learning models to achieve the highest reported Top-1 recall of 88.9% and Top-5 recall of 95.3% across 10 OSS projects.

Existing Limitations: Despite the reported performance metrics, several key factors limit the application in practice of the current state-of-the-art (i.e., VCMatch [54]).

(1) Lack of Representative Training Data: We performed a preliminary study on OSS projects with fixes in GHSA and found that 41.2% of the projects do not include any contributing guidelines.[1]

[1]Evaluating if a CONTRIBUTING.md file exists in a repository.

Without guidelines, contributors submit poor quality commits messages [43]. However, VCMatch (the top-performing prior work) evaluated rigorously maintained projects with restrictive contributing guidelines. For example, FFmpeg's contribution policy mandates that a reference to an issue on the bug tracker is insufficient. Contributors must also include a summary of the bug in the commit message.[2] This dataset is not representative of the broader software supply chain. We address this limitation by curating data from 3,389 advisories across 2,138 different projects.

(2) Non-contiguous Data Sampling: PatchScout and VCMatch use a random sampling technique to build their training and evaluation datasets. Specifically, for every positive VFC, 5,000 other random commits from the code repository are selected as negative samples. This random sampling can have unintended consequences. For example, the time difference between the commit and the associated CVE file date in VCMatch can become emphasized by the model as a discriminating feature and could lead to overestimation of recall. When commits are randomly selected for analysis, there is a tendency to overlook nearby commits to the VFC that would need analysis in a real-world setting. In contrast, contiguously sampling all commits between the window of the reported fixed version and the prior version of commits does not introduce this issue.

As a result, VFCFinder required approximately 24 times less data than PatchScout and 60 times less data than VCMatch for training purposes due to our contiguous sampling approach. The random sampling approach by PatchScout required 3,329,286 unique commit-to-vulnerability pairs and 8,346,669 pairs for VCMatch. The contiguous sampling approach in VFCFinder required only 138,529 commit-to-vulnerability pairs.

(3) Model Complexity and Risk of Overfitting: VCMatch incorporates 100 features, complicating its interpretability and heightening the risk of model overfitting. Prior machine learning research [3, 14] shows more features lead to a higher variance and tend to overfit noisy patterns in the training data, resulting in poor accuracy on new examples. We confirmed this hypothesis of overfitting through empirical evaluation of testing VCMatch on unseen data.

Our System: In this paper, we propose VFCFinder, a novel approach for helping an analyst match a given security advisory to its VFC. Our key intuition is to leverage the fixed version number, which is available for the overwhelming majority of advisories (84%, see Section 4.3). Specifically, we take the window of commits between the fixed version and the prior version to determine the VFC. We empirically found that advisories with fixes contain 94% of VFCs between this window. We then use a combination of five intuitive features to produce a ranked set of five potential VFCs for a given advisory. These features are: (1) the likelihood a commit fixed a vulnerability, (2) the type of vulnerability fixed, (3) the similarity between the commit message and the advisory details, (4) where the commit appeared in the window, and (5) any direct indicators in the commit message (i.e., CVE/GHSA-ID).

The first two features, VFC fix probability and VFC vulnerability type, are generated by fine-tuning the CodeBERT NL-PL model [13]. The semantic similarity between commits and advisory details is generated from sentence embeddings using a pre-trained language model. The final two features, commit location and CVE/GHSA-ID in the message, are statically generated. Finally, these features are fed into an XGBoost model for ranking. We then introduce a contiguous sampling technique that divides the training and testing sets between fixed and prior versions, simulating the approach a human would take to identify a VFC.

We began our work with the goal of automatically backfilling VFCs to security advisories. However, we found that a fully automatic solution inherently introduces unacceptable risk. Adding the incorrect VFC to a security advisory can result in a false sense of security. While a "human-in-the-loop" process will always be required for matching VFCs to their security advisories, our work seeks to provide a *nearly-automatic* method.

Evaluation and Measurement: We evaluate VFCFinder in two ways. First, we construct a representative dataset consisting of the set of all security advisories from the GHSA database with a known patch link: thousands of projects spanning nine programming languages. VFCFinder identifies the correct VFC for a given security advisory 96.6% of the time within the Top-5 ranked commits and 80.0% within the Top-1 ranked commit. For projects with 15 or fewer commits between version releases, VFCFinder identifies the VFC for a given security advisory with a Top-1 recall of 90.9%. In contrast, running VCMatch on our dataset resulted in a Top-1 recall of 44.0% and a Top-5 recall of 70.1%.

Second, we deploy VFCFinder on over 300 randomly selected GHSA advisories without patch links to demonstrate that VFCFinder generalizes beyond our training and testing data. VFCFinder found the missing patch link with a Top-5 recall of 96.1% and a Top-1 recall of 81.2%.

In summary, we make the following key contributions.

- *We propose a security advisory-to-VFC matching approach that generalizes to nine programming languages and thousands of open-source projects.* In contrast to prior work, which uses 100 features [54], our approach only uses five. By using a smaller set of features, we reduce the amount of variance in the resulting model, allowing similar performance of the model across nine languages. Specifically, VCMatch [54] has a 36 percentage point lower Top-1 recall than VFCFinder when evaluated on VFCFinder's dataset, which spans thousands of projects and nine languages. Whereas VFCFinder performs similar to VCMatch when tested on the VCMatch dataset (not included in VFCFinder's training).
- *We propose a new evaluation standard for security advisory-to-VFC matching tools.* Prior work [51, 54] uses a non-contiguous sampling approach for VFC ranking, which overestimates their recall in practice. Our contiguous sampling approach ranges from 24 times less data to 60 times less data than non-contiguous approaches.
- *We deployed VFCFinder to backfill over 300 security advisories in the GitHub Security Advisory database.* GitHub's security team confirmed all of our submitted VFCs and integrated them into the GHSA database.

Availability: VFCFinder is available on GitHub.[3]

[2]https://ffmpeg.org/developer.html#Contributing

[3]https://github.com/s3c2/vfcfinder

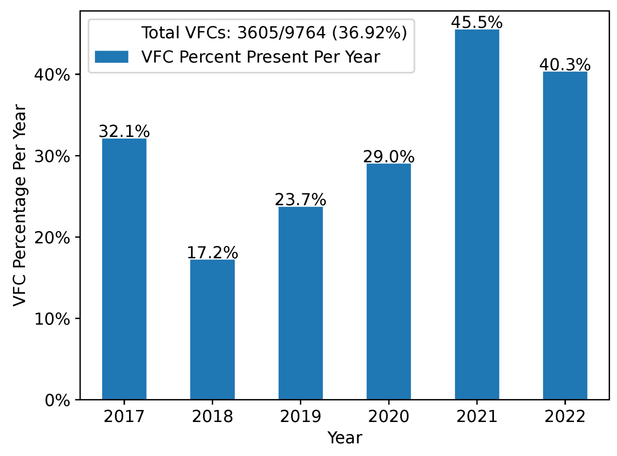

Figure 1: 63.1% of GHSA security advisories are missing their patch link based on a snapshot taken through 2022.

2 BACKGROUND

In the current landscape, vulnerabilities in software are abundant and varied. Effective communication of these vulnerabilities is crucial for the security of the software supply chain. To streamline the reporting and tracking of these vulnerabilities, the U.S. Department of Homeland Security and the Cybersecurity and Infrastructure Security Agency back the MITRE corporation's Common Vulnerability Enumeration (CVE) Program [36]. Through this program, researchers and vendors can report vulnerabilities, which the CVE Assignment Team and CVE Numbering Authorities then review.

MITRE has simplified vulnerability reporting with its dedicated CVE intake form. The form mandates three fields: vulnerability type, the product's vendor, and the affected products with their versions. The form also offers optional sections covering acknowledgment of the vulnerability, attack nature, potential impact, affected components, attack pathways, a recommended description, discovery credits, and relevant references. Once reviewed, the vulnerabilities are assigned CVE IDs and publicly listed on the CVE list [9].

Over the years, various downstream databases have emerged, such as the National Vulnerability Database (NVD)[35], Google's Open Source Vulnerabilities (OSV) [20], and GitHub Security Advisory Database (GHSA)[15], to enrich reports from the CVE list. For example, the NVD offers an additional layer of detail with fields for numerical vulnerability severity scoring, such as the Common Vulnerability Scoring System score. Google introduced OSV and an associated schema emphasizing machine-readable fields. GHSA streamlines updates for vulnerability data, allowing the community to submit enhancement suggestions (e.g., additional reference links) for an associated security advisory. One significant detail within reports is the reference link to the patch link.

Motivating Examples: Figures 2 and 3 present two scenarios on the difficulty of pairing security advisories and VFCs. Figure 2 shows CVE-2019-9721, from the VCMatch dataset, which addressed a denial of service within FFmpeg. The advisory description is clear. The commit message addresses the vulnerability and is similar to the advisory, making it straightforward to pair the advisory to the patch. VCMatch and VFCFinder rank the commit as the top choice.

In contrast, for a cross-site scripting vulnerability in the CKEditor4 project (Figure 3), the CVE description is informative, but the commit message for the patch link contains little useful information. Had the patch link not been included in the original CVE reference links, it would have been nearly impossible for a human to identify. There are 78 commits between the fixed version (4.18.0) and the prior version (4.17.2). VCMatch ranked the corresponding commit as 38th, whereas VFCFinder ranked the commit as third.

Multiple VFCs and Multiple Versions: We anecdotally observed that some security advisories reference multiple VFCs. To understand this relationship, we performed a preliminary study and found that 96% of GHSA security advisories with a fix (see Section 4.3 for data collection process) only list a single VFC. The remaining 4% of advisories offer more than one VFC. Of those, 39% are for patches in multiple versions. For instance, the project *parse-server* from GHSA-2m6g-crv8-p3c6 gives two patch links that correspond to two patched versions (4.10.14 and 5.2.5). In fact, the backport patch link for 4.10.14 needed more changes than the patch for version 5.2.5, demonstrating that it is important to find all VFCs. Therefore, identifying the patches for each reported version is valuable to aid practitioners.

CVE-2019-9721
Description: A denial of service in the subtitle decoder in FFmpeg 3.2 and 4.1 allows attackers to hog the CPU via a crafted video file in Matroska format, because handle_open_brace in libavcodec/htmlsubtitles.c has a complex format argument to sscanf.
Commit Message for VFC: avcodec/htmlsubtitles: Fixes denial of service due to use of sscanf in inner loop for handling braces Fixes: [Semmle Security Reports #19439] Fixes: dos_sscanf2.mkv

Figure 2: An informative commit message for a VFC within FFmpeg for CVE-2019-9721.

CVE-2022-24728
Description: CKEditor4 is an open source what-you-see-is-what-you-get HTML editor. A vulnerability has been discovered in the core HTML processing module and may affect all plugins used by CKEditor 4 prior to version 4.18.0. The vulnerability allows someone to inject malformed HTML bypassing content sanitization, which could result in executing JavaScript code. This problem has been patched in version 4.18.0. There are currently no known workarounds.
Commit Message for VFC: Code refactoring.

Figure 3: A misleading commit message for the VFC within CKEditor4 for CVE-2022-24728.

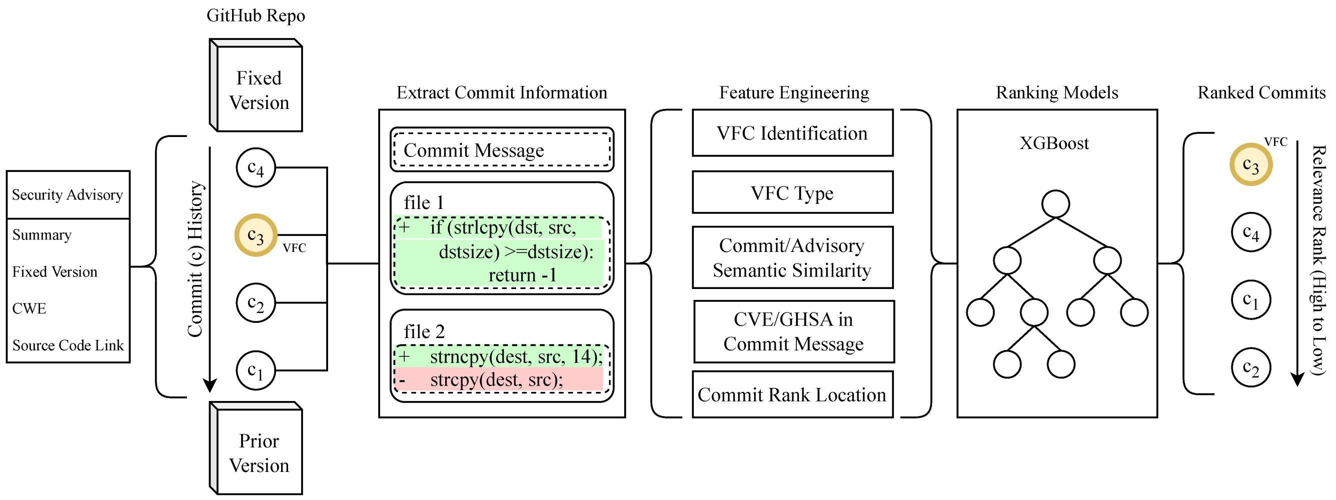

Figure 4: The architecture of VFCFinder ranking commits based on their relevance to fixing a given security advisory.

Commit Window for VFCs: We hypothesized that VFCs generally appear between the reported fixed version and the prior version. To test this hypothesis, we performed a second preliminary study that examined all GHSA security advisories with a VFC and found that around 65% of VFCs appear in this range. An additional 29% of commits are backported from the VFC listed in the security advisory, indicating that there exists a VFC in the hypothesized range. Therefore, 94% of the examined security advisories had the VFC in the expected location. The remaining VFCs not appearing in the anticipated location may be due to unreliable version data [1, 32]. VFCFinder leverages this intuition for its approach.

How the commit window is determined matters. PatchScout's optional branch analysis is the closest to using a commit window: it considers all commits for an entire branch. However, version releases within GitHub are based on git tags, not branches [19]. For example, *parse-server* maintains branches 4.x.x and 5.x.x, each having many minor releases and CVEs. Therefore, the commit window should be based on git tags and not branches. We note that each GHSA advisory contained an exact version [16].

3 THE DESIGN OF VFCFINDER

Figure 4 illustrates the architecture of VFCFinder. The primary goal of VFCFinder is to match security advisories to VFCs. Initially, VFCFinder consumes a security advisory extracting information regarding the fixed and prior version commit windows. Then, considering each commit within the window, the commit message and git diff are extracted. Leveraging CodeBERT [13], VFCFinder generates the first feature, predicting the likelihood that a commit fixed a vulnerability. We fine-tune the CodeBERT model using data from NVD [35], OSV [20], and VulasDB [41]. In addition to VFC identification, VFCFinder uses CodeBERT for vulnerability type classification for each VFC, explicitly focusing on the OWASP Top 10. The third feature is a commit-to-advisory semantic similarity score using SentenceTransformers. The final two features are a CVE or GHSA identifier in commit messages and the commit rank location. These features are fed into a single XGBoost model to create the final ranking of commits relevant to fixing security advisory.

3.1 Extracting Advisory Information

VFCFinder uses the OSV format [6] for security advisories, which provides the following key-value data : (1) a detailed vulnerability summary, (2) CWE type, (3) source code repository, (4) related CVE/GHSA identifiers, and (5) fixed versions. VFCFinder additionally identifies the associated VFC for each fixed version.

Identifying Prior Version and Commit Window: VFCFinder uses git tags, typically used for versioning, to determine the commit window. Once cloning a project locally, all project tags are retrieved (i.e., via `git tag`). The fixed version from the advisory is then matched directly to the tag set. For the prior version tag, VFCFinder uses the package *packaging*,[4] allowing for semantic version sorting of the tags. The tag immediately preceding each fixed version is selected as the prior tag. Upon obtaining the fixed and prior tags, the command `git tag prior_ver...fixed_ver` lists all commits within the specified commit window. As mentioned in Section 2, this approach resulted in a success rate of 94% for identifying the commit window of the provided VFC.

3.2 VFC Identification

The first feature VFCFinder predicts is if a commit resolves a vulnerability, a process based on a fine-tuned CodeBERT model. CodeBERT [13] is a transformer-based architecture [52] equipped with bimodal pre-training for natural language (NL) and programming language (PL). CodeBERT was initially trained on six programming languages paired with function-level documentation. HuggingFace [56] hosts CodeBERT with pre-trained weights, allowing fine-tuning of the model for specific tasks.

We fine-tune CodeBERT for VFC identification using a custom tuning loop, Figure 5, and tuning data described in Section 4.

[4]https://pypi.org/project/packaging/

Tokenization: Before fine-tuning, we transcribe the free-form commit message and code into numerical forms through tokenization. The tokenizer expects two elements from commit data: (a) the commit message, and (b) the git diff featuring modified, deleted, and added code. The tokenizer produces a tensor divided into three sections: *input_ids*, *attention_mask*, and *token_type_ids*. The *input_ids* are a blend of the commit message and git diff as follows: `[CLS] commit_message [SEP] git_diff [EOS]`. Tokens `[CLS][SEP][EOS]` are special separators; `[CLS]` signifies the beginning of the segments, `[SEP]` is a divider between the commit message and raw git diff code, and `[EOS]` is the end-of-sequence token. The *attention_mask* assists the model in identifying *input_ids* padded tokens, indicating which tokens require attention. The *token_type_ids* designates the start and end of sequences, specifically, the length of the commit message tokens and the git diff.

The tokenizer accepts a maximum token count based on the pre-trained model; for CodeBERT, it is 512 tokens. Excess tokens are truncated. In order to minimize truncation, we section the commit data into smaller chunks, each based on a file with changes, and generate tensors from these chunks. This method not only aids in reducing data truncation but also allows us to make predictions and evaluations for individual programming languages separately.

Fine-Tuning: We implemented a classification fine-tuning loop for the CodeBERT model. The model includes an embedding layer that maps input tokens to 768-dimensional vectors and 12 encoder layers. These encoder layers incorporate a self-attention mechanism for focusing on varying parts of the input sequence. Each encoder's intermediate layer executes a non-linear input transformation, followed by a linear output layer transformation. The last encoder layer's output is directed to a pooling layer, averaging the hidden states across the input sequence. This output is then processed through a fully connected layer with an output size of one. During tuning, we use an unweighted binary cross entropy loss function defined as:

$$l_{BCE} = -\left[y \cdot \log x + (1-y) \cdot \log(1-x)\right] \tag{1}$$

where x is the input and y is the target. The logits are passed to a sigmoid activation function, producing the final prediction, ranging from 0 to 1, indicating the VFC positive class probability.

Aggregating Predictions to Commit Level: VFCFinder generates predictions on a per-file basis. This strategy ensures that different programming languages are handled separately during the prediction process. For instance, when a commit updates Python and C files, CodeBERT does not need to process multiple languages simultaneously. Therefore, VFCFinder consolidates file predictions into a total commit prediction. To do so, VFCFinder calculates the arithmetic mean of the file predictions, resulting in a single value between 0 and 1, where 1 suggests a likely vulnerability resolution.

3.3 VFC Vulnerability Type

The fine-tuning for VFC type identification mimics the VFC identification outlined in Section 3.2, differing primarily in the classification tasks. VFC type is categorized based on the OWASP Top 10 and an additional "Other" class that signifies vulnerabilities outside the OWASP Top 10. Initially, we contemplated predicting VFC type at the CWE level, but since MITRE defines 933 different CWE types

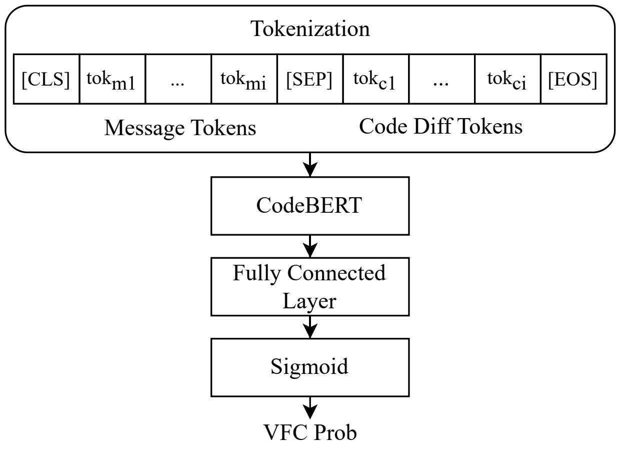

Figure 5: A fine-tuning CodeBERT framework for VFC identification. The fully connected layer of the VFC classification is vector size one. VFC Type identification uses the same framework, but the vector size from the fully connected layer is 10 (i.e., OWASP Top 10) and uses a softmax function.

and the relatively sparse training data, we decided against it. Discussion of VFC type data collection and mapping OWASP Top 10 labels to VFC types is in Section 4.2.

Tokenization: The tokenizer for VFC type is the same for VFC identification, as seen in Section 3.2.

Fine-Tuning: The fine-tuning architecture for VFC type is similar to that of VFC identification. The primary differences are the output size of the fully connected layer, the loss function, and the activation function. The VFC type's output size is 10, denoting its deployment for a 10-class classification task.[5] We specifically use a weighted cross entropy loss function as defined:

$$l_{WCE} = -\, w_y \log \frac{e^{x,y}}{\sum_{c=1}^{10} e^{x,c}} \cdot y \tag{2}$$

where x is the input, y is the target, w is the weight, and c is the number of classes. A softmax activation function is then used on the fully connected output layer, transforming the results into a probability distribution across the classes.

Aggregating Predictions to Commit Level: Predicting the VFC type on a per-file basis requires a distinct commit-level aggregation process. To determine the VFC type, we use the file prediction that has the maximum probability. In specific terms, we use an argmax function to identify and select the OWASP Top 10 type that has the highest probability within a given predictions.

$$argmax f(X) := x : f(s) \leq f(x), \forall s \in X \tag{3}$$

This procedure ensures the selection of the VFC type with maximum confidence. The evaluation of VFC type identification is in Section 5.3.2.

[5]As outlined in Section 4.2, the classification size would be 11, but no examples exist for one of the OWASP Top 10 classes.

3.4 Semantic Similarity

VFCFinder also incorporates the similarity between the commit message and the original advisory. For instance, consider the advisory GHSA-fj7c-vg2v-ccrm and its associated VFC:

> *GHSA-fj7c-vg2v-ccrm description:* "Buffer leak on incoming WebSocket PONG message(s) in Undertow before 2.0.40 and 2.2.10 can lead to memory exhaustion and allow a denial of service."
>
> *undertow@c7e84a0 VFC commit message:* "[UNDERTOW -1935] - buffer leak on incoming websocket PONG message"

VFCFinder uses SentenceTransformers [42], an advanced technique for generating embeddings to produce semantic similarity scores between texts. Specifically, we use the pre-trained all-mpnet-base-v2 model. [6] VFCFinder then feeds these embeddings into a cosine similarity function to identify semantic correlations from the embeddings. The output ranges from -1 (indicating opposite meanings) to 1 (denoting identical meanings). A score of 0 signifies orthogonality or dissimilarity between the two vectors.

Regrettably, not every advisory and VFC commit message is as descriptive as the previous instance. Take the advisory GHSA-rgp5-m2pq-3fmg and the related VFC as an example:

> *GHSA-rgp5-m2pq-3fmg description:* "microweber prior to version 1.2.11 is vulnerable to cross-site scripting"
>
> *microweber@f7f5d41 VFC commit message:* "update"

In the initial example, the cosine similarity score is 0.88, reflecting considerable similarity. However, for the second example, despite being the VFC for the advisory, the cosine similarity score is -0.01.

3.5 Static Features

VFCFinder also incorporates two static features to enhance the classification. We initially considered other static features, similar to those in prior work [21, 51, 54], however, most demonstrated limited feature importance, leading us to retain the following two prominent static features.

CVE/GHSA Identifier: In some cases, developers mention the CVE or GHSA identifiers for advisories directly in commit messages. Naturally, VFCFinder should encapsulate this information. The presence of the CVE/GHSA-ID within the commit message is determined using a direct search method. This feature is encoded as a binary value, with 1 signifying a match.

Normalized Commit Rank Location: In our feature engineering, we observed that VFCs often occur toward the end of the commit lifecycle, typically before the next version release. For instance, the GHSA-prrh-qvhf-x788 advisory resolved a vulnerability across 32 commits, with the VFC (314456d) as the 31st commit, directly preceding the v5.0.2 release.[7] VFCFinder computes $commit_{rank}/commit_{total}$ for the normalized commit rank location, yielding a location of 31/32 = 0.97 for the cited VFC. According to our ground truth dataset (Section 4), the average normalized commit rank location for VFCs is 0.67. This is intuitive for vulnerability patching practices. As vulnerabilities are identified, the expectation is a new software release with the patch provided shortly after.

[6] https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[7] https://github.com/PrestaShop/productcomments/compare/v5.0.1...v5.0.2

Table 1: VFCFinder's ranking model uses five features.

Features	Description
VFC Probability	Probability distribution of commit fixing a vulnerability
VFC Type Match*	Boolean match between advisory and VFC Type prediction
Commit/Advisory Similarity	Similarity score commit message and advisory report
CVE/GHSA ID in Commit*	Boolean match if CVE/GHSA ID in commit message
Commit Rank Location	Normalized commit rank location of a commit in version lifecycle

* We describe this as five features, but the XGBoost model uses seven features. We split individual features for CVE and GHSA, and split VFC type into Top-1 and Top-5.

3.6 Ranking Commits

The final step in VFCFinder is to use the previously described features, Table 1, to rank the commits relevant to the given security advisory. VFCFinder uses XGBoost [7], an iterative gradient-boosting algorithm that progressively incorporates decision trees while adjusting observation weights based on previous inaccuracies. By combining weak learners and predicting residuals from prior trees, XGBoost uses regularization techniques to optimize performance and mitigate overfitting.

To initially tune the hyperparameters of the XGBoost model, we used hyperopt [4], a Bayesian optimization algorithm. The best results were obtained when the learning rate was set to 0.001, and the decision tree depth in the model was restricted to four. Furthermore, we set the maximum number of boosting rounds, i.e., the number of decision trees included in the model, to 1,500.

A binary logistic objective was used during training, classifying each commit as related or unrelated to the security advisory fix. The model outputs the predicted probabilities for each input to belong to the positive class, which range from 0 (non-match) to 1 (match). This process transforms the task into a classification problem. These probabilities are then ranked to denote the likelihood of each commit fixing a security advisory. Section 5 elaborates on the model's evaluation.

4 DATA COLLECTION

Here, we discuss the training and testing datasets for VFCFinder. The data collection process is organized into three sets, each corresponding to a unique classification model: VFC identification, VFC type identification, and the final XGBoost ranking process. Table 2 provides a summary and the aggregate commit count for each set.

4.1 Vulnerability Fixing Commits

We sourced data from three vulnerability databases: NVD[35], OSV[20], and VulasDB [41]. NVD, operated by NIST, is a primary vulnerability disclosure platform. Google's OSV aggregates data from multiple sources (GHSA, PyPA, RustSec, Global Security Database, and OSS-Fuzz) and primarily targets open-source dependencies. VulasDB manually curated vulnerable commits in Java projects

Table 2: Datasets used for training various aspects of VFCFinder. The dataset size indicates commit count.

Dataset	Objective	Size
VFC Identification	VFC and Non-VFCs	54,858
VFC Types	OWASP Top 10 Labeled VFCs	7,847
GHSA Data	Matching VFCs to advisories	138,529

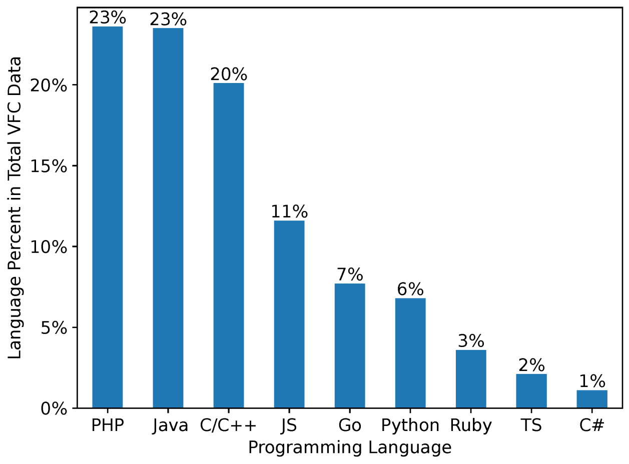

Figure 6: Language breakdown across Vulnerability Fixing Commits within the VFC Identification dataset

and has been used extensively in previous studies [30, 33, 45].

Our training data included 54,858 commits, with 9,143 VFCs and 45,715 non-VFC commits. The commits span nine languages. We started data collection by downloading all datasets, beginning with the NVD, which contained 193,250 CVEs as of August 16, 2022. CVEs contain reference links tagged as a *patch*, typically signifying VFCs. We focused specifically on GitHub commit links, identifying 8,951 GitHub commit patch links. A similar methodology on OSV yielded 3,633 GitHub-referencing VFCs, and VulasDB provided 1,282 Java commits linked to open-source dependencies. After consolidation and de-duplication, 9,143 unique VFCs were obtained.

We then cloned each repository containing the commit and confirmed the commits used languages in C/C++, PHP, Java, JavaScript, Python, Go, Ruby, TypeScript, or C#. To pull the data, we developed an additional tool for parsing commit data, *PatchParser*.[8] The tool pulls key features from GitHub commits and formats them in a way ideal for ML/AI applications. Figure 6 demonstrates the language distribution for the study, emphasizing PHP and Java.

Collecting Non-VFCs: Training requires non-VFCs in addition to VFCs. Consistent with prior work, we keep a ratio of five non-VFCs to one VFC [45]. For every VFC in a repository, we gather five non-overlapping non-VFCs from the same repository.

To collect non-VFCs, we follow methods established in previous studies [33, 45]. Firstly, we collect unique repositories from the VFCs, a total of 2,658. From these repositories, we compile the commit history. We then run a modified version of git-vuln-finder [8], which includes additional keywords from SPI [67], on the commit history. Commits not matched by git-vuln-finder are assumed to be non-VFCs. We then check if the commit modified at least one file associated with the study's target languages; if not, the commit is discarded. After verifying the commits, we shuffle the non-VFCs and match each VFC with five non-VFCs from the same repository. For validation, we randomly sampled 100 non-VFCs to ensure they weren't related to security fixes. This process yielded 45,715 non-VFCs from 2,658 unique repositories.

[8]This tool has been publicly released on PyPI: https://pypi.org/project/patchparser/

Table 3: VFC OWASP Top 10 Distribution

Category	VFC Count
A01: Broken Access Control	1,333
A02: Cryptographic Failures	126
A03: Injection	2,249
A04: Insecure Design	232
A05: Security Misconfiguration	125
A06: Vulnerable and Outdated Components	0
A07: Identification and Authentication Failures	322
A08: Software and Data Integrity Failures	209
A09: Security Logging and Monitoring Failures	30
A10: Sever-Side Request Forgery (SSRF)	88
Other (Weaknesses outside OWASP Top 10)	3,133

4.2 Vulnerability Fixing Commit Types

Security advisories are associated with a common weakness enumeration (CWE), denoting the type of vulnerability. With 933 existing CWE types [27], predicting a VFC's corresponding vulnerability type is challenging. We leverage the CWE to OWASP Top 10 mapping provided by MITRE [27], simplifying our prediction classes. VFCs not in the OWASP Top 10 are categorized as "Other."

We excluded advisories with multiple CWEs. Our preliminary analysis found that less than 2% of the total advisories list multiple CWEs. Furthermore, 1,296 VFCs did not possess a CWE label, resulting in a dataset of 7,847 VFCs with a CWE label. Table 3 details the commit distribution per OWASP Top 10 label and the "Other" class. This imbalance mirrors the real-world vulnerability distribution. Notably, "Vulnerable and Outdated Component" was not a classification within the VFC dataset.

4.3 GHSA Data

We used GHSA advisories to train the final ranking model of VFCFinder. The GHSA data is a subset of the OSV data described in Section 4.1. We selected GHSA as the full pipeline training because it (1) is guaranteed to be on GitHub and (2) is one of the popular advisory databases for open-source projects. An advisory was considered for training if it contained a GitHub code repository link and an identified fixed version. In total, 15.7% of GHSA advisories did not contain fixes, leaving approximately 84.3% of advisories with fixes. We identified 3,605 advisories with a VFC, each containing a fixed version. As Section 2 mentioned, we matched 94% of the VFCs to their associated fixed version range. Therefore, we only considered those VFCs within that range.

Our total dataset consisted of 3,389 advisories, across 2,138 projects, and 138,529 associated commits. The average number of commits between fixed and prior versions was 15. Commit labels were determined based on their association with an advisory, with VFCs in the advisory receiving a label of 1 and others labeled as 0.

Contiguous Data Sampling: The contiguous aspect is to obtain all of the commits in the order in which they appear in the commit lifecycle between the prior and fixed versions. As discussed previously, current state-of-the-art [51, 54] uses a non-contiguous data sampling technique, selecting non-associated commits randomly throughout the project. By narrowing the dataset to commits within the fixed version's window, we eliminate the risk of overemphasizing the time difference between commits and CVE file dates.

FixFinder [21] uses a contiguous sampling approach, but the boundaries are set without respect for the prior and fixed version, creating a selection of commits two years before and one hundred days after the CVE file date for each evaluation, resulting in 2,753,058 commits for their training datasets. VCMatch collected a dataset containing 1,669 vulnerabilities from 10 open-source software projects for training their prediction models. VCMatch used the corresponding fixing commits for each of these vulnerabilities as positive samples. To construct negative samples, they followed the method used in the study of PatchScout, which involved randomly sampling 5,000 other commits in the code repository as negative samples for each positive sample. VCMatch required a total training set of 8,346,669 pairs of vulnerabilities and commits. Each of those approaches requires significantly more data than VFCFinder.

Additionally, our approach separates training and testing datasets to keep advisories distinct and ensures that commit lifecycles within each set are non-overlapping. This strategy ensures the integrity of our training and testing sets, preventing any associated commits from being split between them. Further details on training and testing can be found in Section 5.1.

5 EVALUATION

This section presents the evaluation of VFCFinder on the datasets from Section 4. We pose four research questions:

RQ1: *What is VFCFinder's effectiveness in pairing security advisories and vulnerability fixing commits?* This question assesses the full VFCFinder ranking pipeline. We benchmark VFCFinder against VCMatch [54] on their dataset and across our dataset, representing the software supply chain.

RQ2: *How well does VFCFinder identify VFCs?* We evaluate against nine programming languages for identifying VFCs.

RQ3: *How effective is VFCFinder in determining the VFC type?* Extending past VFC identification, we evaluate how VFCFinder can identify the vulnerability type fixed during the VFC. We classify based on the OWASP Top 10 and an "Other" class.

RQ4: *What features are important for matching security advisories to VFCs?* In addition, we provide insight into how the features of matching security advisories to VFCs impact the output of VFCFinder.

5.1 Evaluation Setup

Our evaluation depends on three datasets: GHSA commits (Section 4.3, RQ1), VFC/Non-VFC labels (Section 4.1, RQ2), and VFC types (Section 4.2, RQ3). These are real-world, up-to-date data from maintained vulnerability databases.

We create a holdout set of 10% of each dataset, preserving vulnerability type and language imbalances through stratified sampling. We apply a 5-fold cross-validation for model fine-tuning on the remaining 90% of data. Each fold results in a model that we test on the holdout set. We then average the model probabilities to create the final holdout set prediction. We confirmed that the training/testing and holdout data do not contain any forms of overlap, which would result in data leakage. Throughout, we use a machine with an Intel i7-9700k CPU, 32GB RAM, and an NVIDIA RTX 3090 Ti GPU.

During the evaluation of VFCFinder, we use Top-N metrics as those in prior work [51, 54]. The Top-N metric measures how often the correct item (or one of the correct items) appears in the top N recommendations or predictions made by a model.

Table 4: A Top-N recall comparison of VCMatch [54] vs VFCFinder on VCMatch's dataset (8.3M commits across 10 OSS projects) and VFCFinder's dataset (138K commits across 2,138 projects). VCMatch's performance on unseen data (VFCFinder data) indicates overfitting, while VFCFinder demonstrates robust performance on new unseen data.

	VCMatch		**VFCFinder**		**Difference***	
Dataset (# cmts)	**Top-1**	**Top-5**	**Top-1**	**Top-5**	**Top-1**	**Top-5**
VCMatch (8.3M)	89.6%	94.3%	81.9%	97.3%	-7.7	+3.0
VFCFinder (138K)	44.0%	70.0%	80.0%	96.6%	+36.0	+26.6

* VFCFinder performance minus VCMatch performance.

5.2 Evaluation Results

Baseline Comparison: We focus our comparison on VCMatch [54], as it is the latest and highest reporting metric for advisory to VFC matching. Additionally, we omit PatchScout [51] from our comparative analysis because the source code is not publicly available. VCMatch replicated PatchScout by themselves in their work and demonstrated a 17-percentage-point performance advantage over PatchScout. We attempted to retrain VCMatch on our data but could not get their source code implementation to work. However, we could get the source code and trained models to work that was publicly accessible within Patchmatch [47], the GUI-based implementation of VCMatch. This availability allows for a noise-free, direct comparison. We first validated Patchmatches implementation matches VCMatch to obtain results using their original dataset before comparing its performance with our dataset. We applied VCMatch on the same contiguous data as VFCFinder and used our contiguous sampling to evaluate VFCFinder on VCMatch data.

Table 4 presents the results of VFCFinder compared to VCMatch. VFCFinder significantly outperforms VCMatch in Top-1 recall by 36 percentage points (80.0% vs. 44.0%) on our dataset, demonstrating greater generalizability. Although VCMatch shows a marginal 7.68 percentage point increase in Top-1 recall when evaluated on its dataset (89.6% vs. 81.9%), it suggests overfitting to its specific data. Furthermore, VFCFinder excels in Top-5 recall on both datasets, indicating a broader and more consistent ability to identify vulnerabilities correctly. These performance metrics in both Top-1

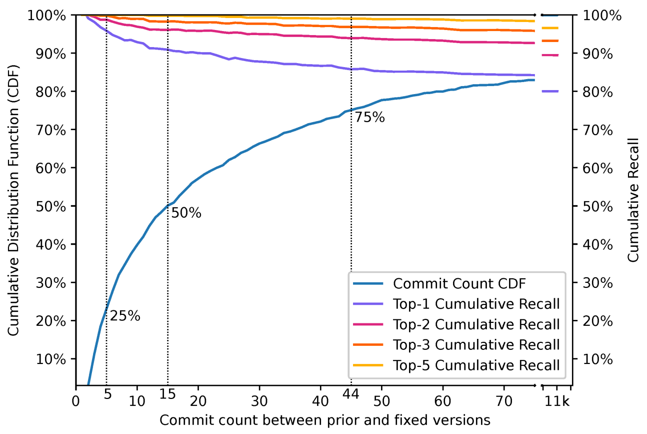

Figure 7: Top-N recall for ranking commits based on the commits between the prior and fixed versions of an advisory. Note, the CDF percentage of the commit count is at 100% at 11,015 commits (the maximum number of commits seen between versions).

and Top-5 recalls validate VFCFinder's robustness in real-world scenarios for matching security advisories to patch links.

Detailed VFCFinder performance: Figure 7 illustrates the performance of VFCFinder. When searching for the vulnerability fixing commits, each version lifecycle will have a different number of commits. We found the median number of commits between versions to be 15. Focusing on the Top-1 recall, when considering the lower quartile (25%) of data, the commit count is less than 5; the recall is 95.9%. This recall value changes to 90.9% for the median (50%, commit count $\leq$ 15) and to 85.8% for the upper quartile (75%, commit count $\leq$ 44). For the entire dataset, the Top-1 recall is 80.0%. For 75% of the data, the Top-2 to Top-5 recall consistently remain above 93%. On the entire dataset, the Top-2, Top-3, and Top-5 recalls are 89.5%, 93.2%, and 96.6%, respectively.

Accurately Ranked Commits: Consider the advisory GHSA-h47x-2j37-fw5m,[9] addressing a critical injection vulnerability in the Infinispan project. This advisory reports two patched versions and provides two VFCs. In the case of version v9.4.17, with 63 commits in the window, VFCFinder correctly ranked the corresponding VFC as first. The VFC identification probability was 0.96, and the model accurately classified the vulnerability type. Despite the final prediction output of 0.30, it was significantly higher than the second-rank commit of 0.08. For the older version, v8.2.12, with just six commits within the window, VFCFinder identified the correct VFC as the top-ranked commit, validating patches for both versions.

Misranked Commits: An instance of an incorrectly ranked commit happened with advisory GHSA-wqv4-9gr3-3qgh related to Jenkins, where VFCFinder ranked the actual VFC seventh among 82 commits. This version had six additional GHSA-IDs, three of which fell into the same OWASP category, leading to the misranking. This highlights the challenge of correctly associating a VFC with its relevant advisory, mainly when multiple vulnerability fixes of a similar type exist between versions.

[9]https://github.com/advisories/GHSA-h47x-2j37-fw5m

Takeaway: Within five commits, VFCFinder produces a 96.6% recall for containing the correct VFC within the prior and fixed versions. Over prior work [54], VFCFinder increases the Top-1 recall by 36% percentage points when applied to various OSS projects.

5.3 VFCFinder Characterization

This section assesses the distinct elements of VFCFinder, with a particular emphasis on the fine-tuned CodeBERT models used for VFC identification and VFC type. Although a substantial body of research exists regarding VFC identification and type (refer to Section 8), they predominantly concentrate on individual facets of VFCFinder. To our knowledge, a comprehensive analysis of VFC identification spanning nine languages has not been extensively explored. Our claim is not to have enhanced VFC identification or VFC type identification; rather, our focus has been to further evaluate CodeBERT across nine languages in VFC identification and VFC type as they are important aspects of VFCFinder.

5.3.1 RQ2: VFC Identification. The VFC identification component of VFCFinder proves effective, achieving an 89.3% macro F1 score and a 94.4% accuracy. Additionally, the performance generalizes across nine languages. We use a base threshold of 0.5 during evaluation to represent a VFC; values below this do not indicate a VFC. Formally,

$$VFC = \begin{cases} 0 & \text{if } \frac{1}{n}\sum_{i=1}^{n} x_i < 0.5, \\ 1 & \text{if } \frac{1}{n}\sum_{i=1}^{n} x_i \geq 0.5 \end{cases} \quad (4)$$

where x is the output from the sigmoid activation function from the fine-tuned CodeBERT model.

VFC Identification Evaluation: Table 5 presents VFC identification results for the dataset outlined in Section 4. Key metrics include a macro F1 score of 89.3%, recall of 87.5%, precision of 91.5%, accuracy of 94.4%, and an area under the ROC curve (AUC) of 95.7%. Weighted F1 on the holdout data is 94.2%. Evaluation against the holdout set yielded 4,452 true negatives, 209 false negatives, 100 false positives, and 704 true positives. The overall performance of VFCFinder is similar to prior work (Section 8) for identifying VFCs, ranging in an F1 score of around 90%.

Correctly Classified VFCs: The true negatives and positives from the model classification within the holdout dataset offer insight into VFCFinder's performance. The average probability for the 4,452 true negatives is 0.04, with a standard deviation 0.07. The true positives average a probability of 0.94, which indicates high model certainty. VFCFinder also applies to less descriptive commit messages, such as d158413 in ckeditor/ckeditor4, to resolve a cross-site scripting vulnerability with a commit message of "Code refactoring" and a probability of 0.84.

Misclassified VFCs: False negatives refer to true VFCs wrongly classified as non-VFCs, while false positives denote non-VFCs incorrectly classified as VFCs. False negatives had a mean probability of 0.18, with a standard deviation of 0.16. For instance, CVE-2017-5553 identified a cross-site scripting (XSS) vulnerability in the b2evolution CMS project, with a single patch link: ce5b36e.[10] The

[10]https://github.com/b2evolution/b2evolution/commit/ce5b36e44b714b18b0bcd34c6db0187b8d13bab8

Table 5: VFC Identification Language Generalization

Language	Macro Precision	Macro Recall	Macro F1
C/C++	92.4%	89.3%	90.7%
Python	90.1%	87.6%	88.8%
TypeScript	86.1%	86.1%	86.1%
JavaScript	89.2%	84.9%	86.9%
PHP	92.9%	88.4%	90.4%
Java	91.0%	84.2%	87.1%
Ruby	93.9%	88.7%	91.0%
C#	87.5%	98.5%	92.1%
Go	89.4%	85.7%	87.4%
Total	91.5%	87.5%	89.3%

Table 6: VFC Type Identification (OWASP-Top 10 + Other Class) Top-N Evaluation

	Precision	Recall	F1	Accuracy
Top-1	80.2%	80.1%	79.7%	80.1%
Top-2	89.3%	88.8%	88.5%	88.8%
Top-3	94.4%	94.3%	94.1%	94.3%
Top-5	98.6%	98.6%	98.6%	98.6%

commit message, *Ignore wrong URLs on markdown plugin*, corresponded to a patch where developers refined an existing regex to accept only URLs beginning with *http://*, *https://*, or */*. The model overlooked this subtle regex adjustment and vague commit message, marking it as a false negative. The probability outputted by VFCFinder was 0.30. Revising the probability thresholds could make the model identify it as a VFC.

False positives produced a mean probability of 0.72 with a standard deviation of 0.16, implying model uncertainty compared to the mean of 0.93 for true positives. For instance, a false positive arose from the Ansible package's bug fix for a missing dependency. Though the commit message, *defend against bad or missing crypt*, initially suggested a vulnerability fix, code review clarified the issue as a failure due to a missing package.

VFC Language Generalization: Table 5 presents the performance metrics for the nine languages in our holdout set. VFCFinder performed well across each programming language. C#, with a 92.1% macro-F1 score, performed best, largely owing to a high recall of 98.5%, despite its lower precision at 87.5%. Interestingly, C# accounted for the smallest training samples in our dataset. TypeScript was the least successful, with an 86.1% macro F1 score.

Takeaway: VFCFinder identifies VFCs with an F1 score of 89.3% and generalizes across nine languages.

5.3.2 RQ3: VFC Type Identification. Table 6 shows the Top-N metrics for VFC vulnerability type classification, labeled as per the OWASP Top 10. VFCFinder scores 80.1%, 88.8%, and 98.6% accuracy for the Top-1, Top-2, and Top-5 labels, respectively. The metrics are weighted to account for data imbalance.

Correctly Classified VFC Types: Figure 8 shows the normalized confusion matrix for predicting vulnerability types, labeled by OWASP Top 10 categories as per Table 3. The matrix's diagonal indicates the recall for each category; for instance, VFCFinder correctly identified 73% of A01: Broken Access Control vulnerability types. In the "Other" class the model correctly detects 84% of them for the Top-1 label. Even with approximately 40% of the data classified as "Other," VFCFinder can accurately distinguish different OWASP classes.

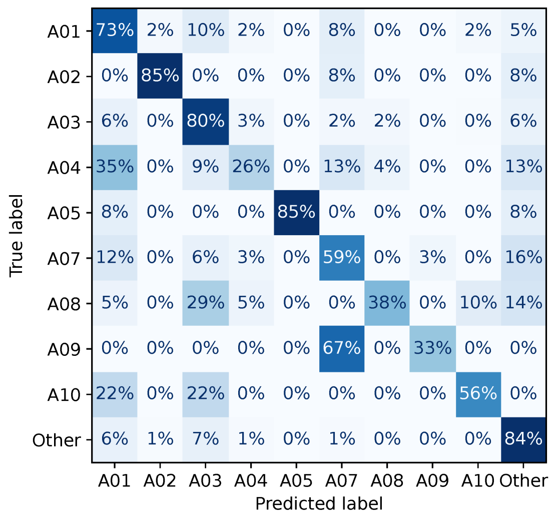

Figure 8: Normalized Confusion Matrix for the VFC Type Top-1 Identification by OWASP Top 10 Categories

Misclassified VFC Types: Figure 8 additionally shows the misclassification analysis of OWASP Top 10 categories. For instance, 22% of A10: SSRF VFCs were predicted as A03: Injection. Despite initial concerns, these types exhibit notable similarities. Taking CVE-2022-1723 as an example, an SSRF was fixed in jgraph/drawio before version 18.0.6, mitigating potential local file access by web server attackers. The commit message "18.0.6 release" fails to specify the patch's purpose. Upon commit review, a *sanitizeUrl(String url)* function emerged to validate URL parameters, a method similar to A03: Injection patching. Thus, SSRF and Injection patches may resemble each other regarding code modification.

Takeaway: VFCFinder correctly classifies the VFC type with a Top-1 accuracy of 80.1% and a Top-5 accuracy of 98.6%.

5.4 Feature Importance

In this section, we explore the impact of five specific features outlined in Table 1 on the performance of VFCFinder. Although machine learning models are often seen as black boxes, using Shapley Additive Explanations (SHAP) values [25] has enhanced our ability to interpret these models. Specifically, SHAP values are used to explain the output of a model. They help to understand how much each feature (or input variable) contributes to the model's prediction, whether positively or negatively. We approach each combination of features as a distinct power set, subjecting each to training. By measuring the contribution of each feature to the

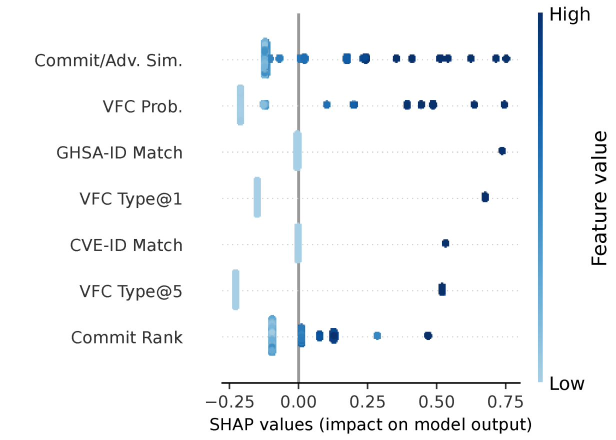

Figure 9: The SHAP values (impact on the model output) of VFCFinder's features. High feature values correspond to high SHAP values, underscoring the importance of all features in VFCFinder's classification. Features are ranked by importance, from highest (Commit/Advisory Similarity) to lowest (Commit Rank).

model's overall predictive outcome, we assess its relative importance. These measured values allow for an understanding of each feature's role in the model.

Figure 9 visually represents the SHAP values distribution across the complete training data pipeline for VFCFinder. The y-axis is the feature value (e.g., a description similarity score of 0.98 would be considered high), with darker colors denoting higher values. The x-axis reflects the aggregated SHAP value. A rise in VFC probability correlates with elevated SHAP values, impacting the model's overall predictive probability. A decrease in VFC probability diminishes the likelihood of a match. Features with higher values, such as VFC probability, commit advisory similarity, and types of vulnerability fixes, are more likely to correspond to an accurate patch link.

Takeaway: Each feature in VFCFinder provides a strong contribution to the matching of security advisories to VFCs.

6 GHSA MISSING LINKS

This section explores an empirical study of VFCFinder applied to a set of GHSAs missing VFC links. As stated in Section 2, around 63% (6,159/9,764) of GHSAs do not have VFC links. We pose the research question: *How does VFCFinder work on existing real-world advisories with missing VFC links?*

6.1 Considered GHSA Advisories

Section 4.3 details VFCFinder prerequisites: a source code link, a fixed version, and a prior vulnerable version, which are not universally available across advisories. Approximately 2,129 (34.6%) of VFC-lacking GHSA advisories do not contain a source code link. As described in the Appendix, we created a simple methodology for obtaining the missing GitHub repository. We can directly locate project links on GitHub using the GHSA package name (e.g., Source Code/Issues/Homepage). This method yielded source code links for approximately 1,092 GHSA advisories missing VFCs. However, 1,037 (16.8%) GHSA advisories continue to lack source code links. We have submitted this data to GitHub.

In total, 5,122 GHSA advisories still lacked VFC links. During repository cloning, 239 (3.88%) did not use tags to define the commit window (Section 3). Not all advisories have been fixed; 1,537 (24.9%) do not contain fixed versions, which would not have a VFC. Three (0.05%) advisories did not contain a previous vulnerable version. Ultimately, VFCFinder could be used on 3,343 advisories.

Table 7: Results based on 334 reviewed GHSA Advsiories missing their VFC. Top-N recall is calculated on the 308 advisories VFCFinder found inside and outside of the Top-5.

GHSA Breakdown		Top-N Recall for Found VFCs	
Total Reviewed	334	Top-1	81.2%
Found in Top-5	296 (88.6%)	Top-2	88.9%
Found outside Top-5	12 (3.59%)	Top-3	93.5%
Could Not Find	25 (7.78%)	Top-5	96.1%

6.2 Missing Link Results

Table 7 provides the results of VFCFinder applied to GHSAs missing VFCs. We assessed VFCFinder using a random 10% (334) subsample of 3,343 advisories. We manually validated the results of VFCFinder to confirm the correct VFC was found.

In total, 334 advisories and VFCFinder's output underwent manual review. VFCFinder's output identified the VFC for 296 (88.6%) advisories. For 12 (3.59%) cases, the patch link was found but not among VFCFinder's Top-5. In 25 (7.78%) cases, the patch link wasn't found in the reported fixed versions, leading us to assume these 25 advisories may have incorrect fixed version data. For VFCFinder's Top-N recall calculation, we considered advisories where the VFC was found or wasn't in the Top-5, resulting in 308 advisories. The Top-N recall results were as follows: 81.2% at Top-1, 88.9% at Top-2, 93.5% at Top-3, and 96.1% at Top-5.

Community Contribution: As a valuable contribution to the community, we submitted all found patch links (308) back to GitHub. The GHSA database welcomes community enhancements to advisories [18]. The security team at GitHub independently reviews the suggested updates to determine if the security advisory will be updated. As discussed in Section 2, advisories can list multiple versions where a vulnerability is fixed. In response to this, for each advisory that mentioned multiple fixed versions, we submitted a VFC corresponding to each of those versions. All 308 patches submitted to GitHub were accepted.

Takeaway: The recall of our empirical study of missing VFCs matches the evaluation in Section 5, demonstrating generalizability. The GitHub security team reviewed and merged all 308 VFCs into GHSA.

6.3 Ethics and Disclosure

Before submitting VFC links to the GHSA database, we consulted our Institutional Review Board (IRB), which confirmed that analyzing public open-source software projects for vulnerabilities does not require IRB approval as it doesn't involve human subject research. The data is public, and our work aims to improve the quality of security advisories for the public.

Initially, we contacted GitHub to see if they would be interested in VFC data. After confirming GitHub's interest in the VFC data and agreeing to manage about 300 updates, we limited our submissions to roughly ten per day to prevent overwhelming the team. Due to the amount of data, the GitHub security team recommended we update through their manual advisory update process, allowing the team to also thoroughly validate and approve our VFC links.

7 THREATS TO VALIDITY

Noisy Data: There is potential for noisy labels within our ground truth data. While we randomly sampled our non-VFC commits to confirm the absence of VFCs from within the set, some could be within the remaining data set. We also trust the original stakeholders provided the correct VFC links in the security advisories. We note that prior research has reported errors in NVD data [11, 32].

Unseen Vulnerability Types: VFCFinder cannot label vulnerability types it has not seen. For example, we had no instances of "Vulnerable and Outdated Components" in our training data.

Advisories without Fixed Versions: VFCFinder works with advisories that have fixed versions. Of the total GHSA dataset, 1,537 (15.7%) advisories do not contain fixed versions. These advisories are assumed not to have been fixed and do not contain a VFC.

Silent Vulnerability Fixes: VFCFinder only works if a known security advisory exists for a project. The purpose of VFCFinder is to pair those known security advisories with their associated VFC. Therefore, the task to find silent vulnerability fixes [12, 29, 49, 61, 62] is significantly different than VFCFinder.

8 RELATED WORK

Vulnerability Fixing Commit Identification: While VFCFinder incorporates identifying a VFC, simply identifying the VFC does not match it to a security advisory. Therefore, the overall task of VFCFinder is significantly different from identifying if a commit fixed an arbitrary vulnerability. However, we extend prior work of identifying vulnerability-fixing commits by evaluating CodeBERT across nine different programming languages, whereas existing works have mainly concentrated on C/C++, Python, or Java. The overall performance of the following prior work for identifying VFCs is equivalent to ours, ranging in an F1 score of around 90%.

Earlier work has used stack-based classifiers [66], support vector machine models [33, 45, 46], and voting algorithms [55] to detect security patches. Zhou et al. [63] create separate classifiers (including CodeBERT) for commit messages and code changes, subsequently integrating the results through a stacking ensemble technique. Building on this foundation, Nguyen et al. [33] incorporated commit issues as an additional feature for classification. Vulcurator [30] extended the model using CodeBERT to analyze messages, issues, and code diffs. Vulcurator reported up to 87% on a Python dataset. Hong et al. [22] consider multiple data sources, including issue trackers like Bugzilla, GitHub projects, and Stack Overflow. TMVDPatch [64] reported a 90% F1 score on a single C/++ dataset using an attention-based BLSTM model that relies on the commit message and the patch to identify VFCs. Midas [31] introduced a multi-granularity approach, focusing exclusively on code to identify vulnerability fixes at line, hunk, and file levels. Zhou et al. [61] introduced CoLeFunDa to identify vulnerability fixes at the function level with an AUC of 80% only on a Java dataset. VFFinder [29] introduced an AST graph-based approach for identifying VFCs based only on code changes. Evaluating against 507 C/C++ projects, VFFinder reported an F1 score of 69%. Zhou et al. [65] introduced CCBERT, a new transformer-based pre-trained model to represent code changes. Within a downstream task of identifying bug-fixing commits, they reported a 91.8% F1 score on a set of Linux bug-fixing patches using just the code. Zuo et al. [68], using a transformer-based architecture relying only on the commit message, reported an F1 score of 89.1% across C/C++ projects with commit patches from NVD. In parallel, Sun et al. [49] confirmed that Codebert with commit messages and code changes provided the best performance in terms of VFC prediction.

Vulnerability Fixing Commit Type: Related to our work has been identifying the type of vulnerability fixed during a commit, but the vast majority has been identifying CWE types for longer descriptions in security advisories [2, 10, 28, 37, 44]. TreeVul [38] uses a CodeBERT to embed the removed and added code during a git diff, which is then fed into a hierarchical Bi-LSTM encoder to predict the CWE type of a VFC. TreeVul reported a 72% weighted F1 score at the depth-3 CWE prediction and up to an 85% F1 score at the depth-1 CWE prediction on 6,541 commits from 1,560 GitHub OSS projects. In addition, CoLeFunDa [61] can categorize the correct CWE type with an F1 score of 50% and AUC of 85%. Contrastingly, DAA [12] took a non-ML approach for VFC identification, which, while capable of producing corresponding CWE types, suffers from recall issues due to reliance on Static Application Security Testing tools. While TreeVul, CoLeFunDa, and DAA are similar to a feature of our work, we predict by the OWASP Top 10.

9 CONCLUSION

The completeness of security advisories is crucial for downstream users, yet about 63% of GitHub Security Advisories lack their patch link. This paper presents VFCFinder, a tool designed to perform security advisory to VFC matching. VFCFinder achieved a recall of 96.6% in identifying the correct VFC within five commits. Our approach demonstrates that a streamlined pipeline and concise features offer superior generalization over complex systems. Applied to GHSA advisories lacking VFCs, VFCFinder found 96.1% of the VFCs within the Top-5. GHSA has accepted and merged over 300 of our submitted VFCs.

ACKNOWLEDGMENTS

This work is supported in part by NSF grants CNS-1946273 and CNS-2207008. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. 2022. Cleaning the NVD: Comprehensive Quality Assessment, Improvements, and Analyses. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2022), 4255–4269. https://doi.org/10.1109/TDSC.2021.3125270

[2] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. 2020. Automation of Vulnerability Classification from its Description using Machine Learning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*. 1–7. https://doi.org/10.1109/ISCC50000.2020.9219568

[3] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* 116, 32, 15849–15854. https://doi.org/doi.org/10.1073/pnas.1903070116

[4] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 115–123. https://proceedings.mlr.press/v28/bergstra13.html

[5] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning* (Bonn, Germany) *(ICML '05)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/1102351.1102363

[6] Oliver Chang and Russ Cox. 2023. Open Source Vulnerability Format. https://ossf.github.io/osv-schema/.

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[8] CVE-Search. 2022. Finding potential software vulnerabilities from git commit messages. https://github.com/cve-search/git-vuln-finder

[9] CVEProject. 2023. CVE cache of the official CVE List in CVE JSON 5.0 format. https://github.com/CVEProject/cvelistV5.

[10] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Pothen, and Ehab Al-Shaer. 2021. V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)* (2021-10). 1–12. https://doi.org/10.1109/DSAA53316.2021.9564227

[11] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 869–885. https://www.usenix.org/conference/usenixsecurity19/presentation/dong

[12] Trevor Dunlap, Seaver Thorn, William Enck, and Bradley Reaves. 2023. Finding Fixed Vulnerabilities with Off-the-Shelf Static Analysis. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. 489–505. https://doi.org/10.1109/EuroSP57164.2023.00036

[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Association for Computational Linguistics*. https://doi.org/10.48550/ARXIV.2002.08155

[14] Stuart Geman, Elie Bienenstock, and René Doursat. 1992. Neural networks and the bias/variance dilemma. *Neural computation* 4, 1 (1992), 1–58. https://doi.org/doi.org/10.1162/neco.1992.4.1.1

[15] GitHub. 2022. GitHub Advisory Database. https://github.com/github/advisory-database

[16] GitHub. 2023. Best practices for writing repository security advisories - Affected Versions. https://docs.github.com/en/code-security/security-advisories/guidance-on-reporting-and-writing-information-about-vulnerabilities/best-practices-for-writing-repository-security-advisories#affected-versions.

[17] GitHub. 2023. GitHub Advisory Database - Contributions. https://github.com/github/advisory-database#contributions.

[18] GitHub. 2023. GitHub Advisory Database - Who reviews the pull requests? https://github.com/github/advisory-database#who-reviews-the-pull-requests.

[19] GitHub. 2023. GitHub Docs - About releases. https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases.

[20] Google. 2022. OSV - Open Source Vulnerabilities. https://github.com/google/osv.dev https://github.com/google/osv.dev.

[21] 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 Trans. Softw. Eng. Methodol.* (mar 2024). https://doi.org/10.1145/3649590 Just Accepted.

[22] Hyunji Hong, Seunghoon Woo, Eunjin Choi, Jihyun Choi, and Heejo Lee. 2022. xVDB: A High-Coverage Approach for Constructing a Vulnerability Database. *IEEE Access* 10 (2022). https://doi.org/10.1109/ACCESS.2022.3197786

[23] Matthieu Jimenez, Yves Le Traon, and Mike Papadakis. 2018. [Engineering Paper] Enabling the Continuous Analysis of Security Vulnerabilities with VulData7. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 56–61. https://doi.org/10.1109/SCAM.2018.00014

[24] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, Dallas, Texas, USA, 2201–2215. https://doi.org/10.1145/3133956.3134072

[25] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[26] Siqi Ma, Ferdian Thung, David Lo, Cong Sun, and Robert H. Deng. 2017. VuRLE: Automatic Vulnerability Detection and Repair by Learning from Examples. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, 229–246.

[27] MITRE. 2022. CWE - CWE-1344: Weaknesses in OWASP Top Ten (2021) (4.9). https://cwe.mitre.org/data/definitions/1344.html.

[28] Sarang Na, Taeeun Kim, and Hwankuk Kim. 2017. A Study on the Classification of Common Vulnerabilities and Exposures using Naïve Bayes. In *Advances on Broad-Band Wireless Computing, Communication and Applications*, Leonard Barolli, Fatos Xhafa, and Kangbin Yim (Eds.). Springer International Publishing, Cham, 657–662.

[29] Son Nguyen, Thanh Trong Vu, and Hieu Dinh Vo. 2023. VFFINDER: A Graph-Based Approach for Automated Silent Vulnerability-Fix Identification. In *2023 15th International Conference on Knowledge and Systems Engineering (KSE)*. 1–6. https://doi.org/10.1109/KSE59128.2023.10299438

[30] 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 (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1726–1730. https://doi.org/10.1145/3540250.3558936

[31] Truong Giang Nguyen, Thanh Le-Cong, Hong Jin Kang, Ratnadira Widyasari, Chengran Yang, Zhipeng Zhao, Bowen Xu, Jiayuan Zhou, Xin Xia, Ahmed E. Hassan, Xuan-Bach D. Le, and David Lo. 2023. Multi-Granularity Detector for Vulnerability Fixes. *IEEE Transactions on Software Engineering* 49, 8 (2023), 4035–4057. https://doi.org/10.1109/TSE.2023.3281275

[32] Viet Hung Nguyen and Fabio Massacci. 2013. The (un)reliability of NVD vulnerable versions data: an empirical experiment on Google Chrome vulnerabilities. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security* (Hangzhou, China) *(ASIA CCS '13)*. Association for Computing Machinery, New York, NY, USA, 493–498. https://doi.org/10.1145/2484313.2484377

[33] 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)*. 51–62. https://doi.org/10.1109/SANER53432.2022.00018

[34] NIS. 2023. Proposal for legislation to improve the UK's cyber resilience. https://www.gov.uk/government/consultations/proposal-for-legislation-to-improve-the-uks-cyber-resilience/proposal-for-legislation-to-improve-the-uks-cyber-resilience.

[35] NIST. 2022. National Vulnerability Database. https://nvd.nist.gov/

[36] NIST. 2023. CVEs and the NVD Process. https://nvd.nist.gov/general/cve-process.

[37] Mengyuan Pan, Po Wu, Yiwei Zou, Chong Ruan, and Tao Zhang. 2023. An automatic vulnerability classification framework based on BiGRU-TextCNN. *Procedia Computer Science* 222 (2023), 377–386. https://doi.org/10.1016/j.procs.2023.08.176 International Neural Network Society Workshop on Deep Learning Innovations and Applications (INNS DLIA 2023).

[38] 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)*. 957–969. https://doi.org/10.1109/ICSE48619.2023.00088

[39] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. 2022. Vuln4Real: A Methodology for Counting Actually Vulnerable Dependencies. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1592–1609. https://doi.org/10.1109/TSE.2020.3025443

[40] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. 2020. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering* 25, 5, 3175–3215. https://doi.org/doi.org/10.1007/s10664-020-09830-x

[41] Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. 2019. A Manually-Curated Dataset of Fixes to Vulnerabilities of Open-Source Software. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 383–387. https://doi.org/10.1109/MSR.2019.00064

[42] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings

using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[43] Sofia Reis, Rui Abreu, and Corina Pasareanu. 2023. Are security commit messages informative? Not enough!. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (Oulu, Finland) *(EASE '23)*. Association for Computing Machinery, New York, NY, USA, 196–199. https://doi.org/10.1145/3593434.3593481

[44] Jukka Ruohonen and Ville Leppänen. 2018. Toward Validation of Textual Information Retrieval Techniques for Software Weaknesses. In *Database and Expert Systems Applications*, Mourad Elloumi, Michael Granitzer, Abdelkader Hameurlain, Christin Seifert, Benno Stein, A Min Tjoa, and Roland Wagner (Eds.). Springer International Publishing, Cham, 265–277.

[45] Antonino Sabetta and Michele Bezzi. 2018. A Practical Approach to the Automatic Classification of Security-Relevant Commits. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 579–582. https://doi.org/10.1109/ICSME.2018.00058 ISSN: 2576-3148.

[46] Arthur D Sawadogo, Tegawendé F Bissyandé, Naouel Moha, Kevin Allix, Jacques Klein, Li Li, and Yves Le Traon. 2022. SSPCatcher: Learning to catch security patches. *Empirical Software Engineering* 27, 6, 151. https://doi.org/doi.org/10.1007/s10664-022-10168-9

[47] Kedi Shen, Yun Zhang, Lingfeng Bao, Zhiyuan Wan, Zhuorong Li, and Minghui Wu. 2023. Patchmatch: A Tool for Locating Patches of Open Source Project Vulnerabilities. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 175–179. https://doi.org/10.1109/ICSE-Companion58688.2023.00049

[48] Snyk. 2023. How are patches validated? https://support.snyk.io/hc/en-us/articles/360000925338-How-are-patches-validated-.

[49] 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)*. 970–982. https://doi.org/10.1109/ICSE48619.2023.00089

[50] Qing Sun, Lili Xu, Yang Xiao, Feng Li, He Su, Yiming Liu, Hongyun Huang, and Wei Huo. 2022. VERJava: Vulnerable Version Identification for Java OSS with a Two-Stage Analysis. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 329–339. https://doi.org/10.1109/ICSME55016.2022.00037

[51] 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. https://doi.org/10.1145/3460120.3484593

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[53] Shu Wang, Xinda Wang, Kun Sun, Sushil Jajodia, Haining Wang, and Qi Li. 2023. GraphSPD: Graph-Based Security Patch Detection with Enriched Code Semantics. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2409–2426. https://doi.org/10.1109/SP46215.2023.10179479

[54] Shichao Wang, Yun Zhang, Liagfeng 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)*. 589–600. https://doi.org/10.1109/SANER53432.2022.00076

[55] X. Wang, K. Sun, A. Batcheller, and S. Jajodia. 2019. Detecting "0-Day" Vulnerability: An Empirical Study of Secret Security Patch in OSS. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 485–492. https://doi.org/10.1109/DSN.2019.00056 ISSN: 1530-0889.

[56] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, Online, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[57] Chunqiu Steven Xia and Lingming Zhang. 2023. Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT. *arXiv preprint arXiv:2304.00385* (2023).

[58] 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* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 860–871. https://doi.org/10.1145/3540250.3549125

[59] Shalanda Young. 2022. Enhancing the Security of the Software Supply Chain through Secure Software Development Practices. https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf.

[60] Hang Zhang and Zhiyun Qian. 2018. Precise and Accurate Patch Presence Test for Binaries. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 887–902. https://www.usenix.org/conference/usenixsecurity18/presentation/zhang-hang

[61] 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)*. 2565–2577. https://doi.org/10.1109/ICSE48619.2023.00214

[62] 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)*. 705–716. https://doi.org/10.1109/ASE51524.2021.9678720 ISSN: 2643-1572.

[63] 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)*. 705–716. https://doi.org/10.1109/ASE51524.2021.9678720

[64] Xin Zhou, Jianmin Pang, Zheng Shan, Feng Yue, Fudong Liu, Jinlong Xu, Junchao Wang, Wenfu Liu, and Guangming Liu. 2023. TMVDPatch: A Trusted Multi-View Decision System for Security Patch Identification. *Applied Sciences* 13, 6 (2023). https://doi.org/10.3390/app13063938

[65] Xin Zhou, Bowen Xu, DongGyun Han, Zhou Yang, Junda He, and David Lo. 2023. CCBERT: Self-Supervised Code Change Representation Learning. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 182–193. https://doi.org/10.1109/ICSME58846.2023.00028

[66] 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 (ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 914–919. https://doi.org/10.1145/3106237.3117771

[67] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. 2021. SPI: Automated Identification of Security Patches via Commits. *ACM Transactions on Software Engineering and Methodology* 31, 1, 13:1–13:27. https://doi.org/10.1145/3468854

[68] Fei Zuo, Xin Zhang, Yuqi Song, Junghwan Rhee, and Jicheng Fu. 2023. Commit Message Can Help: Security Patch Detection in Open Source Software via Transformer. In *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*. 345–351. https://doi.org/10.1109/SERA57763.2023.10197730

A APPENDIX

A.1 Process for identifying the missing source code links

Each ecosystem contains an online registry (e.g., PyPI -> https://pypi.org/). Using the package name from a GHSA Advisory, we can do a direct lookup in the respective online registry for the package project links (e.g., Source Code/Issues/Homepage) that point to GitHub.

PyPI Example:

(1) Example GHSA-ID: GHSA-m6xf-fq7q-8743
(2) We extract package name: bleach
(3) We then try to parse the project links on the respective online registry: https://pypi.org/project/bleach/
(4) We extract the homepage from the online registry -> https://github.com/mozilla/bleach
(5) We then return the link that points to a GitHub Repository

Maven Example: Maven based projects were not so simple. The following steps were followed to identify Maven source code links:

(1) First, we search for the project using the following API (https://search.maven.org/solrsearch/select?q={groupId}+AND+a:{artifactId}&rows=10&wt=json)

(2) We extract the package name from the GHSA object: org.springframework.security:spring-security-core
(3) We search using the following API: https://search.maven.org/solrsearch/select?q=org.springframework.security+AND+a:spring-security-core&rows=10&wt=json
(4) We match based on the groupId and artifactID parsed from the package name.
(5) We pull the latest version of the package from the. Example response:
   (a) Latest Version: 6.0.1
(6) We pull the POM file for the latest version using the following API https://search.maven.org/remotecontent?filepath=org/springframework/security/spring-security-core/6.0.1/spring-security-core-6.0.1.pom
(7) We then search the POM file for the SCM tag that points to a GitHub repository:
   (a) <connection>scm:git:git://github.com/spring-projects/spring-security.git</connection>

Our process obtained 56% of the missing source code links. We provided the appropriate source code to the GitHub security team to pull these links for their security advisories.