5-2023

CHRONOS: Time-aware zero-shot identification of libraries from vulnerability reports

Yunbo LYU

Thanh Le CONG

Hong Jin KANG

Ratnadira WIDYASARI

Zhipeng ZHAO

See next page for additional authors

CHRONOS: Time-Aware Zero-Shot Identification of Libraries from Vulnerability Reports

Yunbo Lyu*§, Thanh Le-Cong*§, Hong Jin Kang*, Ratnadira Widyasari*,
Zhipeng Zhao*, Xuan-Bach D. Le†, Ming Li‡, David Lo*
*Singapore Management University †The University of Melbourne ‡Nanjing University
{yunbolyu, tlecong}@smu.edu.sg, {hjkang.2018, ratnadiraw.2020}@phdcs.smu.edu.sg, zpzhao@smu.edu.sg,
bach.le@unimelb.edu.au, lim@nju.edu.cn, davidlo@smu.edu.sg

***Abstract*—Tools that alert developers about library vulnerabilities depend on accurate, up-to-date vulnerability databases which are maintained by security researchers. These databases record the libraries related to each vulnerability. However, the vulnerability reports may not explicitly list every library and human analysis is required to determine all the relevant libraries. Human analysis may be slow and expensive, which motivates the need for automated approaches. Researchers and practitioners have proposed to automatically identify libraries from vulnerability reports using extreme multi-label learning (XML).**

While state-of-the-art XML techniques showed promising performance, their experimental settings do not practically fit what happens in reality. Previous studies randomly split the vulnerability reports data for training and testing their models without considering the chronological order of the reports. This may unduly train the models on chronologically newer reports while testing the models on chronologically older ones. However, in practice, one often receives chronologically new reports, which may be related to previously unseen libraries. Under this practical setting, we observe that the performance of current XML techniques declines substantially, e.g., F1 decreased from 0.7 to 0.28 under experiments without and with consideration of chronological order of vulnerability reports.

We propose a practical library identification approach, namely CHRONOS, based on zero-shot learning. The novelty of CHRONOS is three-fold. First, CHRONOS fits into the practical pipeline by considering the chronological order of vulnerability reports. Second, CHRONOS enriches the data of the vulnerability descriptions and labels using a carefully designed data enhancement step. Third, CHRONOS exploits the temporal ordering of the vulnerability reports using a cache to prioritize prediction of versions of libraries that recently had reports of vulnerabilities.

In our experiments, CHRONOS achieves an average F1-score of 0.75, 2.7x better than the best XML-based approach. Data enhancement and the time-aware adjustment improve CHRONOS over the vanilla zero-shot learning model by 27% in average F1.

***Index Terms*—zero-shot learning, library identification, unseen labels, extreme multi-label classification, vulnerability reports**

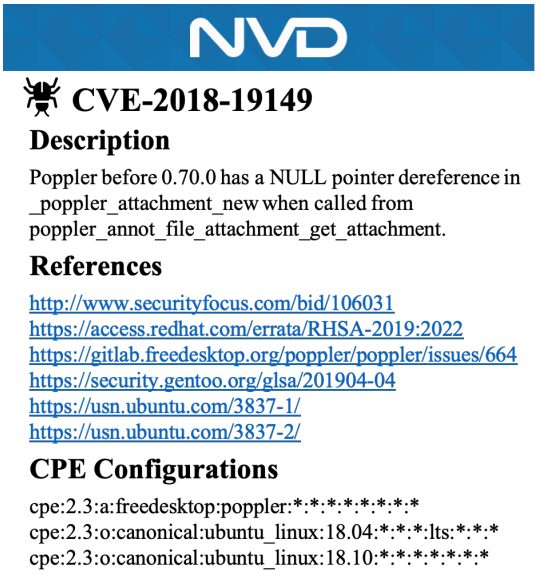
I. INTRODUCTION

The use of third-party libraries is commonplace in software development, however, software engineers have to be aware of and manage library vulnerabilities [1]–[3]. Software Composition Analysis tools have been proposed to assist developers by warning them of vulnerable libraries included in a software project's dependencies. These tools, including those built by industrial companies, such as Veracode [4] and Snyk [5], are now widely deployed but depend on an up-to-date and accurate vulnerability database. These databases indicate libraries, known vulnerabilities, vulnerable library versions, and other data [4]. The database is maintained by security researchers who, through their domain knowledge and manual effort, curate vulnerability reports from multiple sources, including the National Vulnerability Database (NVD). A vulnerability report has an identification number, a CVE (Common Vulnerability Enumeration) ID and a description of the vulnerability. While a CPE (Common Platform Enumeration) configuration indicates a package or library that is related to the vulnerability, this configuration is not exhaustive [6]. Hence, security researchers have to annotate each vulnerability report with the affected libraries and even specific versions if they think the versions are noteworthy. For alerting developers, these databases require a mapping between each vulnerability ID and the libraries (and specific versions) that are affected by the vulnerability. For example, Figure 1 shows the vulnerability report of CVE-2018-19149. While the vulnerability report mentions "Poppler", other software systems such as "evince" and "okular" are also affected [7].

There is usually a delay from vulnerability disclosure (i.e., publicly posted on NVD and assigned a CVE ID) to developers updating their dependencies [8], [9]. This motivates automated approaches that speed up the work of security researchers. For curating vulnerabilities to update vulnerability databases, Chen et al. from Veracode have proposed to automatically identify libraries from the vulnerability reports [6]. The study has formulated the problem as an extreme multi-label classification (XML) problem [6]. Characterized by the sparsity of the data and the large space of possible labels, XML problems are challenging for standard machine learning approaches. Recently, Haryono et al. [10] found that the most effective XML approach for library identification is a deep learning-based approach, LightXML [11].

While XML techniques were shown to be effective in the experiments of prior studies [6], [10], we observe that there are practical concerns that need to be addressed. Every year, new libraries are included in the NVD. If an XML approach is trained strictly on data prior to the inclusion of the new library, it would not produce the correct labels as output. In other words, existing library identification approaches will fail

§Equal contribution



CVE-2018-19149

Description

Poppler before 0.70.0 has a NULL pointer dereference in _poppler_attachment_new when called from poppler_annot_file_attachment_get_attachment.

References

http://www.securityfocus.com/bid/106031
https://access.redhat.com/errata/RHSA-2019:2022
https://gitlab.freedesktop.org/poppler/poppler/issues/664
https://security.gentoo.org/glsa/201904-04
https://usn.ubuntu.com/3837-1/
https://usn.ubuntu.com/3837-2/

CPE Configurations

cpe:2.3:a:freedesktop:poppler:*:*:*:*:*:*:*:*
cpe:2.3:o:canonical:ubuntu_linux:18.04:*:*:*:lts:*:*:*
cpe:2.3:o:canonical:ubuntu_linux:18.10:*:*:*:*:*:*:*

Fig. 1. NVD entry for CVE-2018-19149. Each vulnerability report has a description, some references, and CPE configurations. While "evince" is affected by the vulnerability, the term "evince" does not appear in the report.

to predict previously unseen libraries.

We performed an empirical study of the number of new libraries with vulnerabilities each year. Our analysis indicates that up to 70% libraries associated up to 50.7% of vulnerability reports each year cannot be correctly identified by the previously proposed approaches [6], [10]. As the training dataset would not contain any NVD entries related to the libraries, the XML techniques would not correctly identify vulnerabilities related to these libraries. To address this practical concern, we reformulate the library identification task as a generalized zero-shot learning task. We split the dataset chronologically; as new libraries may be added to the NVD, there would be libraries in the testing dataset that do not correspond to any NVD entry in the training dataset.

To tackle the aforementioned task, we propose a practical library identification approach namely CHRONOS based on zero-shot XML, that is capable of predicting previously unseen labels from vulnerability reports. To achieve this, CHRONOS relies on two main observations: (1) Additional documents referenced in the NVD entry, e.g., bug reports, mailing lists, can help distinguish multiple previously unseen labels from one another. (2) Exploiting temporal connection between vulnerability reports and affected libraries can help boost the prediction accuracy. The key intuition is that if a vulnerability was reported for a particular version of a library recently, it is likely that new vulnerabilities will be reported for the same version rather than for an older version.

Towards this end, CHRONOS implements several techniques to retrieve and process additional sources of information referenced from NVD entries, enriching the vulnerability description with more information. To exploit temporal connection between vulnerability reports and affected libraries, CHRONOS uses a cache to track the libraries related to the most recently seen NVD entries. The cache enables a reranking of CHRONOS's predictions by favouring libraries and versions that were most recently observed to be vulnerable.

In our experiments, CHRONOS achieves an average F1 of 0.75, outperforming the LightXML [11] approach by 167.9% in average F1 (0.75 to 0.28). This demonstrates the superior performance of CHRONOS over traditional non zero-shot XML models. Compared to a manually handcrafted approach that directly matches library names against the vulnerability description, CHRONOS performs 92.3% better.

Compared to an approach using only the CPE, CHRONOS performs 3 times better. Our analysis reveals that each component of CHRONOS contributes positively to its effectiveness. Removing the data enhancement step reduces performance by 6.7%. Removing the time-aware adjustment reduces performance by 9.2%. Overall, CHRONOS improves over a vanilla zero-shot XML model by 27% (0.75 vs. 0.59).

Our study has practical and research significance. It helps in securing the software supply chain by automating slow manual analysis, and highlights practical concerns, such as the chronological order of data, in developing automated tools.

In summary, our paper makes the following contributions:

- **Problem reformulation.** We reformulate the task of predicting libraries to consider the reports chronologically based on publication dates. The task is a generalized zero-shot extreme multi-label (XML) classification task; vulnerability reports in the testing dataset may be related to libraries that did not appear in the training dataset.
- **Approach.** We propose CHRONOS, a zero-shot learning technique. CHRONOS uses data enrichment and a time-aware adjustment step to favour more recently seen versions of each library. CHRONOS's dataset [12] and implementation [13] are publicly available.
- **Experiments.** We evaluate CHRONOS and show that CHRONOS outperforms the strongest previously proposed approach by 167.9% in average F1 on the realistic but more challenging experimental setting.

The rest of this paper is organized as follows. Section II covers the background of our work. Section III formulates the task. Section IV introduces CHRONOS. Section V discusses our experimental results. Section VI presents a deeper analysis of our findings and threats to validity. Finally, Section VIII concludes the paper.

II. BACKGROUND

A. Extreme Multi-label Classification for Identifying Libraries

Extreme Multi-label Learning (XML) models assign relevant labels to documents [11], [14]. Each document may be assigned multiple labels. Tasks employing XML techniques are characterized by an extremely large label space and sparse data. XML approaches have to select a small subset of relevant labels out of *millions of possible labels*. Moreover, many labels have only a few instances associated with them, posing a challenge for standard machine learning techniques [11], [14].

Chen et al. [6] from Veracode, a well-known application security company, formulated the task of identifying libraries

affected by vulnerabilities given the vulnerability report. Their experiments revealed that the NVD report's CVE configuration was insufficient for identifying *every* affected library. Their experiments highlighted the promise of applying XML techniques for the task. Each vulnerability report may describe multiple affected libraries, and the space of all libraries is enormous. These characteristics present challenges for traditional Machine Learning techniques but are addressed by XML techniques. A recent study by Haryono et al. [10] assessed recent XML techniques on the task. Their experiments revealed that the deep learning-based approach, LightXML [11], led to the greatest increase in performance among recently proposed XML techniques.

We show that while the powerful XML techniques had strong performance in the experiments of prior studies, the experiments did not capture every practical consideration. In this study, we reformulate the task as a generalized zero-shot learning problem.

B. Generalized Zero-Shot Learning

The challenge of predicting labels that do not appear during training is established in the machine learning literature [15]. In zero-shot learning, the training and testing labels are disjoint. In generalized zero-shot learning, both seen and unseen labels appear in the testing dataset.

For generalized zero-shot XML problems, ZestXML [16] has been previously proposed. ZestXML aims to exploit the sparsity of the data in XML tasks. During training, ZestXML learns to project a small number of features to be close to the features of the relevant labels. Using a novel optimization technique based on the assumption that only a few features are relevant to a label, ZestXML is able to be trained quickly. We use ZestXML in our approach as it is targeted at zero-shot learning tasks; ZestXML can output labels without any training data as long as the document features closely match the label features (c.f., Section IV-B).

III. Problem Formulation

A. Usage Scenario

A security researcher is monitoring and curating vulnerability data from multiple sources, including the National Vulnerability Database (NVD). For each vulnerability report, the researcher has to map it to a set of relevant libraries. Without an automated tool, the security researcher has to rely only on their domain knowledge and carefully analyze the vulnerability description and references to mailing lists/bug reports. Unfortunately, there is a large number of vulnerability reports and many possible libraries, and even each libraries may have many different versions. As a result, human analysis is slow and may be error-prone. An automated approach that predicts relevant libraries would augment the manual analysis performed by the researcher.

B. Problem Formulation

In this work, following prior works [6], [10], we formulate the problem of library identification from vulnerability reports as an XML problem, where vulnerability reports and possible libraries, which can be enumerated from package managers (e.g., npm and pypi), are considered documents to be classified and their labels, respectively. If security researchers believe that particular versions of the libraries are noteworthy [6], the vulnerability report may be labelled with specific versions of the affected library (e.g. the standard library of java 1.7 vs java 1.8). Different from prior works, we reformulate the problem in the zero-shot setting as follows.

Prior Knowledge. A labelled dataset $\mathcal{D} = (\mathcal{V}, \mathcal{L}, \mathcal{M})$, where $\mathcal{V}$ is set of vulnerability reports and $\mathcal{L}$ is set of labels for $\mathcal{V}$ is a mapping from $\mathcal{V}$ to set of subsets of $\mathcal{L}$, where $\mathcal{M}(v) \subseteq \mathcal{L}$ is the set of labels for a vulnerability report v. A label $l \in \mathcal{L}$, may identify a particular library version.

Input. A new (unlabelled) dataset $\mathcal{D}_{new} = (\mathcal{V}_{new}, \mathcal{L}_{new})$ where $\mathcal{V}_{new} \neq \mathcal{V}$ is set of new vulnerability reports and $\mathcal{L}_{new} \supseteq \mathcal{L}$ is set of labels.

Output. A mapping $\mathcal{M}_{new}$ from $\mathcal{V}_{new}$ to set of subsets of $\mathcal{L}_{new}$ such that $\mathcal{M}_{new}(v) \subseteq \mathcal{L}_{new}$ is the set of labels for each vulnerability report $v \in \mathcal{V}_{new}$.

The approaches proposed in prior studies [6], [10] are built upon the assumption that the set of labels in the labelled (training) dataset $\mathcal{L}_{new}$ are identical to the labels in new (testing) dataset $\mathcal{L}$. More formally,

$$\mathcal{L}_{new} = \mathcal{L} \tag{1}$$

The previous studies treat the problem as a supervised learning task. As a result, they trained supervised learning models such as LightXML [11] or FastXML [14] to learn the mapping $\mathcal{M}$ from labelled dataset and then use the trained model as the mapping $\mathcal{M}_{new}$. Unfortunately, in a practical setting, the assumption (1) does not hold.

In this paper, we reformulate the problem of library identification from vulnerability reports to consider the possibility of unseen libraries. We relax assumption (1) to:

$$\mathcal{L}_{new} \supseteq \mathcal{L} \tag{2}$$

This assumption means that $\mathcal{L}$, i.e., the set of *seen labels* belonging to the labelled dataset, should be a subset of $\mathcal{L}_{new}$, i.e., the set of *all seen and unseen labels*. This relaxation allows our problem formulation to include unseen labels and be more suitable in practical settings.

IV. Proposed Approach

Figure 2 illustrates the overall framework of CHRONOS. CHRONOS identifies libraries for each vulnerability report. There are three main components in CHRONOS: (1) data enhancement, (2) a zero-shot learning XML model, and (3) time-aware adjustment.

The first component, data enhancement (Section IV-A), addresses the lack of discriminating information to identify links between the vulnerability description and possible labels. This enriches vulnerability descriptions by collecting data from

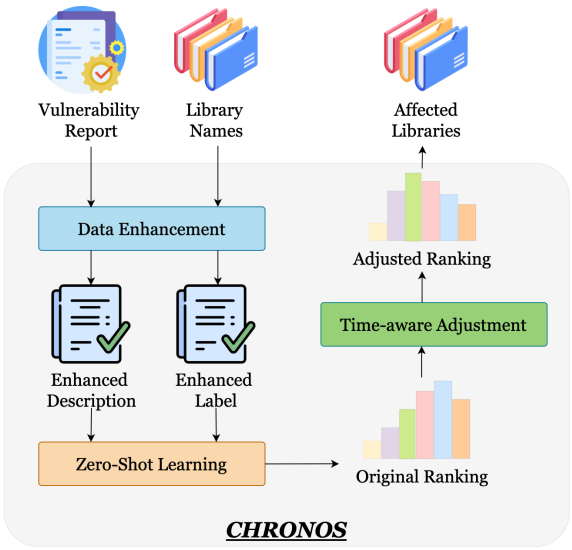


Fig. 2. The overview of CHRONOS

their website references (Section IV-A1) and then performing preprocessing (Section IV-A2) to clean the descriptions. CHRONOS also enriches the label features by splitting labels to sub-words (section IV-A3). The second component, a zero-shot XML model (Section IV-B), uses a ZestXML model to answer the question, "How likely is a vulnerability description and a library name to be relevant?" using the features from vulnerability descriptions and the label features. The third component, time-aware adjustment (Section IV-C), exploits the temporal ordering of the vulnerability reports using a cache to prioritize predictions of versions of libraries that were more recently affected by vulnerabilities.

A. Data Enhancement

1) Collecting Reference Data: A vulnerability report can come with a list of website references. These website references can include any pertinent references (e.g., solutions, workarounds, etc). This information may be helpful to identify affected libraries. Before a vulnerability report is passed to the zero-shot XML model, CHRONOS fetches the web references from the vulnerability report to extract the aforementioned textual information.

Unfortunately, there are many different domains (e.g. 1,054 unique domains on our dataset) for the web references. This is a challenge for web scrapping tools as each domain has a unique web page layout. To address this problem, we only extract data from the highly frequent domains in our dataset as shown in Table I, which cover 82.3% of the vulnerability reports in the dataset. For each web reference, CHRONOS automatically crawls references of depth 1, i.e., references explicitly linked from the report. In total, we crawled 28,783 references. As the quality of data collected is more important than its quantity, our crawler accommodates the different website structures of each domain to fetch the vulnerabilities' details. We crawl each reference's page title and description of the vulnerability, which

TABLE I
MOST FREQUENT DOMAINS WITH THEIR NUMBER OF OCCURRENCES

Domain	#Occurrences
access.redhat.com	6383
list.opensuse.org	4882
github.com	3479
debian.org	2853
oracle.com	2717
securitytracker.com	2286
security.gentoo.org	1875
ubuntu.com	1752
usn.ubuntu.com	1687
openwall.com	1517
lists.fedoraproject.org	1299
bugzilla.redhat.com	1225

may name affected products (e.g. CVE-2014-1512 references https://ubuntu.com/security/notices/USN-2151-1, which indicates that "Thunderbird" is affected).

2) Preprocessing Reference Data: We preprocess the documents collected in the previous step in order to clean the data before using it for training CHRONOS's model. We perform the following three steps:

- In the first *basic* preprocessing step, we remove non-alphanumeric characters. This preprocessing step is done using the regular expression: *"[a-zA-Z][a-z]+"*.
- We perform stemming and stopwords removal [17]. This is performed automatically using the spaCy [18] package.
- We remove $x\%$ of words sorted by the number of occurrences in reference data since they are common words that will increase the noise. We also remove words that appear more than y times in a single reference. Some words appear frequently in the collected reference data. As they are not specific to a particular CVE or libraries, they can be removed. Removing these words was previously found to be effective for identifying libraries [6]. x and y are parameters that are tuned on the validation dataset.

Finally, CHRONOS merges the processed reference data with each vulnerability description to produce the final description, d, from the vulnerability reports.

3) Library Sub-word Splitting: Next, the label processing component of CHRONOS enriches the features that help determine labels associated with vulnerability reports, i.e., libraries, denoted as $\mathcal{L}$. To maximize the chance of matching a label against mentions of the library in the vulnerability description, CHRONOS initializes the label features with different forms of the library name. Library names usually comprises several words. While the words in some library names are visually separated based on existing conventions to ease reading (e.g. org.apache.tika), not all libraries have names with clear conventions for splitting them (e.g. org.**spring**framework, py**openssl**). Moreover, when considering libraries related to vulnerabilities, the library names adhere to different conventions as the libraries come from a variety of different languages and ecosystems [6]. Therefore, we split each library name into its constituent subtokens using a suitable library name splitter.

The decision of whether to decompose the library names into subtokens, i.e. morphological units, has important im-

plications. With subtokens, the number of features increase, making the selection the most important features of each label more difficult. On the other hand, breaking up a library name into multiple units has advantages as CHRONOS may be able to identify more valuable features. Subtokens are more common than the original library name, enabling CHRONOS to find more connections between unseen and seen library names, which improves CHRONOS to predict unseen labels.

For splitting tokens into subtokens, several approaches use Mining Software Repositories techniques, such as LINSEN [19], Samurai [20], Spiral [21]. We apply the state-of-the-art Spiral token splitter [21], which was shown to be more effective than other methods [21].

B. Zero-shot Learning

The next component in CHRONOS is a zero-shot learning model, which takes the vulnerability reports as input and produces a list of labels and their intermediate relevance scores as output. Particularly, CHRONOS employs the current state-of-the-art technique, ZestXML [16], as the core machine learning model. Following the original paper [16], CHRONOS uses TF-IDF [22] to extract the feature vectors for the descriptions and labels. ZestXML takes input as the extracted features models the relevance between descriptions and labels by analyzing their linear feature interactions. Particularly, given a description d and a label l, the relevance score between are calculated as follows.

$$R(d,l) = d^\top \mathbf{W} l \tag{3}$$

where a large (small) $R(d,l)$ value means high (low) relevance between d and l. $d^\top$ is the transpose vector of d. For simplicity, ZestXML are absorbed into Equation (3) by appending a constant feature to d and l.

$\mathbf{W}$ is $\mathbb{R}^{D'} \times \mathbb{R}^{L'}$ a matrix of model parameters, which is learned to correctly classify all description and label pairs in the training dataset (D' and L' indicate their high dimensional, sparse TF-IDF feature vectors of descriptions and labels). Particularly, ZestXML learns the model parameters W via a regularized logistic regression as follows.

$$\begin{aligned} &\min_{\mathbf{W}} \frac{1}{2}\|\mathbf{W}\|_L^2 + \lambda \sum_{i=1}^{D}\sum_{j=1}^{L} \log\left(1 + e^{-y_{ij} d_i^\top \mathbf{W} l_j}\right) \\ &\text{s.t., } \|\mathbf{W}_{i*}\|_0 \le K \ \forall i \in \{1, \cdots, D'\} \end{aligned} \tag{4}$$

where D, L are the number of descriptions and labels. K, λ are hyper-parameters of the model and $\|\mathbf{W}_{i*}\|_0$ is the number of non-zeros in the i th row of $\mathbf{W}$. y_{ij} is the ground truth relevance between the description i and the label j, where $y_{ij} \in \{-1, 1\}$ and $y_{ij} = 1$ denotes that description i is relevant to the label j.

To determine the optimal model parameters W for the aforementioned training objective, ZestXML proposed an extension of Hard Thresholding Pursuit [16] termed XHTP. ZestXML was designed with the assumption that the features of each label is sparse. Similar to other second-order optimization algorithms, an iteration of XHTP consists of 2 successive steps: (1) approximation in which XHTP approximates the training objective in Equation (4) by a quadratic form and minimizes it to obtain a sparsified solution, and (2) refinement, in which XHTP refines the values of non-zero parameters to fit the original objective better. However, unlike other second-order optimization algorithms, XHTP exploits assumptions such as feature independence and a favorable starting point, i.e., $\mathbf{W} = 0$ to achieve highly sparsified and accurate model parameters in just one iteration of approximation and refinement steps. In this way, ZestXML improves its efficiency.

C. Time-aware Adjustment

Algorithm 1 Time-aware adjustment that favours new library versions and recently observed labels

Require:
- $\mathcal{L}_{highest} \leftarrow$ top-i most relevant labels for each description
- version_store $\leftarrow$ a map of a label to newer versions of the same library
- cache $\leftarrow$ recently seen labels
- $R(d,l) \leftarrow$ a relevance score between a description, d and a label, l
- $f \leftarrow$ an update function. Given in Equation 5

```
function TIME-AWARE ADJUSTMENT(L_highest)
    for l ∈ L_highest do
        FAVORNEWVERSION(l, version_store, cache)
    end for
    for l ∈ L_highest do
        R(d,l) ← f(R(d,l))
    end for
end function
```

Algorithm 2 Transferring the relevance scores from old to new versions of the same library

Require:
- $l \leftarrow$ a label
- version_store $\leftarrow$ a map of a label to newer versions of the same library
- cache $\leftarrow$ recently seen labels
- $R(d,l) \leftarrow$ a relevance score between a description, d and a label, l

```
function FAVORNEWVERSION(l, version_store, cache)
    for l^new ∈ version_store[l] do
        if l^new ∈ cache and R(d,l^new) > R(d,l) then
            R(d,l^new) ← max(R(d,l^new), R(d,l))
            R(d,l) ← 0
            break
        end if
    end for
end function
```

Next, given the labels and their relevance scores from the zero-shot XML model, the time-aware adjustment component modifies the relevance scores. We observe that vulnerabilities

in the same time range are more likely to affect the same versions of the libraries. Thus, CHRONOS uses a strategy to prioritize versions of libraries that have been recently affected by vulnerabilities. In Algorithm 1, CHRONOS's time-aware adjustment component uses two steps to modify the relevance scores: favor newer library versions (lines 2–4) and add a recency bias (lines 5–7).

These two steps use a version store and a cache. The **version store** tracks the different versions of each library. Given a version of a library, the version store returns the list of labels corresponding to newer versions of the library, sorted in descending order by their versions (i.e., newest versions first).

The **cache** stores the recently affected libraries using a Least Recently Used (LRU) replacement policy with a cache size c. Chronologically, as vulnerability reports are labelled with their true labels (e.g. as a security researcher annotates the ground-truth on each report after considering the predictions of CHRONOS), CHRONOS adds the label into the cache. When a new label is added while the cache is full, the new label replaces the oldest entry in the cache.

For each description d, ZestXML models ranks the labels l in set of library $\mathcal{L}_{highest}$ via the relevance scores $R(d, l)$. CHRONOS uses the cache to modify the $R(d, l)$ at prediction time through two successive steps: replacement and update. For efficiency, CHRONOS considers only the top-i highest relevant labels. i is a parameter which is tuned on the validation dataset and will be discussed in Section V-A.

The time-aware adjustment **favors newer library versions** by replacing the old versions of a library in the top-i highest relevant labels by a newer version if certain conditions are satisfied. As seen in Algorithm 2, if a newer version, l^{new}, of a library is in the cache and they have smaller $R(d, l^{new})$ values (line 3), i.e., $R(d, l^{new}) < R(d, l)$, CHRONOS will set the relevance of the new label to be $R(d, l)$ (line 4) and remove the old versions from consideration (line 5).

The time-aware adjustment has a **recency bias** and uses the cache to modify the top-i highest $R(d, l)$ values. The update function f (used in Algorithm 1 on line 6) is formulated as:

$$f(R(d,l)) = \begin{cases} R(d,l) + \alpha \times \bar{R} & l \in cache \\ R(d,l) & l \notin cache \end{cases} \quad (5)$$

where the magnitude of α is determined by the relative recency of l in the cache. More recently observed libraries are more likely to be the label of a vulnerability report. The parameter values require careful selection. If α and $\bar{R}$ are too big, the adjustment function dominates the predictions of CHRONOS. Conversely, if they are too small, they do not affect the final scores. α and $\bar{R}$ are defined as follows:

$$\alpha = \frac{M}{L_{recency} + 1} \quad (6)$$

$$\bar{R} = \frac{\sum_{j=1}^{i} R(d, l_j)}{i} \quad (7)$$

where M determines the magnitude of favouring recently vulnerable library versions. $L_{recency}$ is the relative recency for label l in the cache, which ranges from 0 to $c - 1$. 0 implies that the label was just added, while a recency of $c - 1$ implies that the label is the least recently used label in the cache. $\bar{R}$ is the average of top-i $R(d, l)$ values. The values of M and i are tuned on the validation dataset.

TABLE II
PARAMETERS USED FOR LIGHTXML

Parameter	Value
Learning rate	0.0001
Epoch	30
Batch size	4
SWA warmup	10
SWA step	200
Feature	Transformer generated vectors

TABLE III
PARAMETERS USED FOR CHRONOS

Parameter	Value
cache size (c)	300
ranking-related factor (M)	8
update range (i)	10
x	50
y	15

V. EVALUATION

A. Implementation details

We implement CHRONOS and the baseline approaches using the PyTorch library and Python. The models are trained and evaluated on a Docker environment running Ubuntu 18.04 with Intel(R) i7-10700K @ 3.8GHz, 64GB RAM, and 2 NVIDIA RTX 2080 Ti GPU (11GB of graphics memory for each). For tuning LightXML's hyper-parameters, we run on AMD EPYC 7643 @ 2.3GHz, 512GB, and 4 RTX A5000.

The detailed hyper-parameters of LightXML and CHRONOS are shown in Table II and III, respectively. We run LightXML and CHRONOS 5 times. These parameters are tuned through a grid search on the validation dataset considering the following possible values: c is in {100, 200, 300, 400}, M is in {0.5, 1, 2, 4, 8, 16, 32}, i is in {5, 10, 50, 100}, x is in {50, 60, 70, 80, 90, 95}, and y is in {5, 6, 7, 8, 9, 10, 13, 15}. The details of the grid search for LightXML are provided in the replication package [13]. LightXML produced slightly different results with a standard deviation of 0.005, while CHRONOS produced the same results each time. As the results are stable, we find that it is not necessary to repeat the experiments more than 5 times as our findings will not change even with more runs.

B. Dataset

To evaluate effectiveness of our approach, we use a dataset of 7,665 vulnerability reports with 4,682 labels from the NVD

TABLE IV
THE STATISTICS OF TRAINING, VALIDATION AND TESTING DATASET

Dataset	#Vulnerability Reports	#Labels
Training	3111	1378
Validation	1814	1094
Testing	2740	1432

(National Vulnerability Database) and SCA (Software Composition Analysis) vulnerability database, initially collected by Chen et al [6]. Each report comprises a unique CVE ID, its vulnerability description, a list of web references, its CPE (Common Platform Enumeration) configuration, and its labels (i.e., the affected libraries). For a fair comparison, we use the same preprocessing steps done by Haryono et al. [10]. Each vulnerability report is a single document after applying these preprocessing steps:

- **Description**: Non-alphanumeric characters and non-noun words are removed. Words that appear in more than 30% of the vulnerability data are removed.
- **References**: Non-alphanumeric characters are replaced with whitespace.
- **CPE configuration**: Possible library names are retrieved using a regular expression based on the CPE format [23].

Finally, we have a dataset of 7,665 vulnerability reports with 2,817 labels. We split the dataset chronologically into training/validation/testing datasets. Our dataset comprises vulnerability reports published in a span of six years (2014-2019). The training/validation/testing splits follow the ratio 3:1:2. Particularly, vulnerability reports from years of 2014-2016, 2017 and 2018-2019 form the training, validation, and testing dataset respectively. Table IV shows each dataset in detail.

C. Experimental Metrics

Following previous works [6], [10], we evaluate the effectiveness of CHRONOS and three baselines in terms of Precision (P), Recall (R) and F1-score (F1) calculated for the top-k prediction results with k=1,2,3. These metrics are standard metrics for the evaluation of XML tasks in prior studies [6], [10], [11]. Particularly, for each technique, we obtain their prediction score for the possible labels of a given vulnerability report and then rank the labels based on the score to obtain the top-k prediction.

Given a top-k prediction $lb\_k(v)$ and the actual labels $\hat{lb}(v)$ for a given vulnerability report v, $P@k$ and $R@k$ are defined as follows:

$$P@k(v) = \frac{lb\_k(v) \cap \hat{lb}(v)}{min(k, |\hat{lb}(v)|)} \quad R@k(v) = \frac{lb\_k(v) \cap \hat{lb}(v)}{|\hat{lb}(v)|}$$

We normalize $P@k$ to compare each approach against an ideal approach. For example, in our dataset, 60.58% of vulnerability reports are assigned only one label. For these reports, without normalization – the min() expression in the denominator – the maximum achievable P@3 is 0.33. The normalized P@k formula above considers this best possible score. The results for original $P@k$ is presented in Appendix A.

Then, we compute the average of the precision and recall calculated above to obtain the $P@k$ and $R@k$ that we use to compare the performance between CHRONOS and three baselines (n refers to the number of labels):

$$P@k = \frac{1}{n}\sum_{v=1}^{n} P@k(v) \quad R@k = \frac{1}{n}\sum_{v=1}^{n} R@k(v)$$

Finally, we compute $F1@k$, which is the harmonic mean of $P@k$ and $R@k$.

$$F1@k = 2 \times \frac{P@k \times R@k}{P@k + R@k}$$

D. Baseline Approaches

To assess CHRONOS, we use the following baselines:

CPE Matcher: CPE Matcher is a simple baseline proposed by Chen et al. [6]. CPE matcher uses the libraries listed in the CPE configuration of a vulnerability report. Particularly, CPE matcher retrieves library names and versions from the CPE configurations and outputs them as the labels on the vulnerability report.

Traditional IR. We use TF-IDF with bag-of-ngrams ($n \leq 2$) to obtain feature vectors for the vulnerability reports and the labels. For each report, the cosine similarity between its feature vector and every label is computed. The top ranked labels are selected as output.

Exact Matcher: As simple approaches can sometimes outperform complex ones in software engineering tasks [24]–[26], we propose a handcrafted heuristic-based approach that we term an Exact Matcher, which directly matches labels to their occurrences in vulnerability reports. Exact Matcher ranks the label based on the number of occurrences that the library name occurs in each vulnerability report and outputs the top-k labels that occurs most frequently.

LightXML: LightXML [11] is the best-performing XML technique on the library identification problem with non zero-shot setting in the experiments by Haryono et al. [10]. LightXML is a deep learning-based XML technique that uses transformer-based models with dynamic negative sampling. Particularly, LightXML divides labels into clusters based on balance K-Means [27] and represent vulnerability reports using dense 768-dimensional vectors obtained from transformer-based models such as RoBERTa [28], BERT [29] and XL-Net [30]. LightXML uses generative cooperative networks with dynamic negative label sampling to score all label clusters and returns possible libraries. Finally, LightXML scores every returned label and outputs the top-k highest score labels.

E. Research Questions

We aim to answer the following research questions:

RQ1: *What is percentage of unseen libraries in practice?* This research question investigates the percentage of unseen libraries that do not belong to the training dataset in practice. To answer this question, we investigate the percentage of seen and unseen libraries on our dataset as described in section V-B. Particularly, we count the number of seen and unseen libraries for each year from 2015 to 2019. We consider a library of a vulnerability report as an *unseen label* if it does not appear in vulnerabilities from the training dataset, which includes vulnerabilities reports published chronologically before the reports in the testing dataset.

RQ2: *Is* CHRONOS *effective in identifying libraries from vulnerability reports?* This research question concerns the ability of CHRONOS in identifying libraries from vulnerability

TABLE V
THE STATISTICS OF SEEN AND UNSEEN LIBRARIES PER YEAR DURING THE PERIOD 2015-2019

Year	#Total	#Seen Libraries	#Unseen Libraries
2015	656	312 (47.6%)	344 (52.4%)
2016	896	345 (38.5%)	551 (61.5%)
2017	1094	329 (30.0%)	725 (70.0%)
2018	1094	451 (41.2%)	643 (58.8%)
2019	651	313 (46.5%)	338 (53.5%)

TABLE VI
THE STATISTICS OF VULNERABILITY REPORTS CONTAINING SEEN AND UNSEEN LABELS PER YEAR DURING THE PERIOD 2015-2019. THE #SEEN, #FULLUNSEEN AND #PARTIALUNSEEN DENOTES THE NUMBER OF VULNERABILITY REPORTS ARE RELATED TO ONLY SEEN LABELS, ONLY UNSEEN LABELS AND BOTH SEEN AND UNSEEN ONES, RESPECTIVELY.

Year	#Total	#Seen	#FullUnseen	#PartialUnseen
2015	981	551 (56.2%)	292 (29.8%)	430 (43.8%)
2016	1347	704 (52.3%)	447 (33.8%)	643 (47.7%)
2017	1814	896 (49.3%)	837 (33.2%)	918 (50.7%)
2018	1718	872 (49.5%)	640 (46.1%)	846 (50.5%)
2019	1022	498 (48.7%)	458 (44.8%)	525 (51.3%)

reports. To evaluate our approach, we evaluate CHRONOS on a dataset of 7,665 real-world vulnerability reports in terms of Precision, Recall, and F1-score as described in section V-C. We compare our approach to multiple baselines, including the state-of-the-art technique, LightXML [10], the CPE Matcher [31] and a handcrafted exact matching algorithm. We run each tool five times and report the average results.

RQ3: *Which components of* CHRONOS *contributes to its performance?* CHRONOS contains multiple components, including the data enhancement and time-aware adjustment. In this research question, we investigate the contribution of each component in an ablation study.

F. RQ1: Percentage of Unseen Labels per Year

We investigate the percentage of seen and unseen labels in our dataset. We count the number of seen and unseen labels and their associated vulnerability reports for each year from 2015 to 2019. As 2014 is the first year of our dataset, we exclude it from the table. We consider a label unseen if it does not appear in previous years. The results are reported in Table V and VI.

As shown in Table V, the percentage of unseen labels ranges from 52.4% to 70% during 2015-2019. In particular, unseen labels account for almost half of all vulnerable labels in every years, and the percentage of unseen labels is 70% in 2017.

Concerning the percentage of descriptions associated with unseen labels, Table VI shows that 43.8% to 51.3% vulnerability descriptions during 2015-2019 contain at least one unseen label. Moreover, there are up to 46.1% vulnerability descriptions containing only unseen labels. These results reveal the limitations of a non zero-shot learning techniques on the problem as they cannot correctly predict unseen labels.

Answer to RQ1: Up to 70% of labels are unseen labels. This affects 43.8% to 51.3% of vulnerability descriptions per year. This suggests that existing approaches cannot correctly produce the right labels for half of all vulnerability reports each year.

G. RQ2: Comparison with Baselines

We compare CHRONOS against the baselines approaches with respect to Precision, Recall, and F1 at top-k predictions (k=1,2,3). The detailed results are shown in Table VII.

Table VII shows that CHRONOS achieves an F1 of 0.75 on average with the F1@1 of 0.67, F1@2 of 0.77 and F1@3 of 0.80. These results indicate that CHRONOS consistently outperforms the baseline tools, outperforming the best baseline by 131.1%, 83.3%, 70.2%, and 92.3% in terms of F1@1, F1@2, F1@3, and average F1 respectively. Compared to LightXML, CHRONOS outperforms it by 167.9% in average F1. Notably, LightXML underperforms the Exact Matching baseline in every metric. This highlights the challenge of the zero-shot experimental setting as LightXML was the best-performing approach in the experiments of the prior study [10]

When considering only either Precision or Recall, CHRONOS is still the best performing approach. On Precision, CHRONOS is outperforms the best baseline by 127.3%, 81.8%, and 70.8% in the top-1, top-2, and top-3 predictions respectively. On Recall, CHRONOS outperforms the best baseline by 134.6%, 82.9% and 71.7% in the top-1, top-2, and top-3 predictions respectively.

Compared to ZestXML alone, CHRONOS improves by 27% in average F1. The improvements come from both increases in precision (up to 33.9%) and recall (35.5%). This highlights the contributions of the domain-specific components, i.e. data enhancement and time-aware adjustment.

Answer to RQ2: Yes, CHRONOS is 92.3% better in average F1 compared to the strongest baseline. The improvements come from both increases in precision (up to 127.3%) and recall (up to 134.6%).

H. RQ3: Ablation Study

In this experiment, we evaluate the relative contribution of two components, data enhancement and time-aware adjustment, to the overall performance of our approach, CHRONOS. Table VII shows the results of our experiments.

As shown in Table VII, removing each component reduces the overall performance of CHRONOS. The performance of CHRONOS drops in every metric. The average F1 of CHRONOS without data enhancement and time-aware adjustment are declined from 0.75 to 0.7 ($\downarrow$ 6.7%) and 0.65 ($\downarrow$ 9.2%), respectively. This suggests that both data enhancement and the time-aware adjustment are crucial to the effectiveness of CHRONOS. Moreover, the results also suggest that the time-aware adjustment is more essential to CHRONOS.

TABLE VII
COMPARISON OF THE EFFECTIVENESS OF CHRONOS WITH THE STATE-OF-THE-ART TECHNIQUES. THE BOLD NUMBERS DENOTE THE BEST RESULTS FOR EACH METRIC.CHRONOS W/O DE, CHRONOS W/O TA DENOTES THE RESULTS OF CHRONOS WITHOUT DATA ENHANCEMENT AND TIME-AWARE ADJUSTMENT, RESPECTIVELY.

Model	P@1	R@1	F1@1	P@2	R@2	F1@2	P@3	R@3	F1@3	Avg. F1
Exact Matching	0.33	0.26	0.29	0.44	0.41	0.42	0.48	0.46	0.47	0.39
CPE Matcher	0.27	0.26	0.26	-	-	-	-	-	-	-
Traditional IR	0.34	0.25	0.29	0.36	0.33	0.35	0.41	0.39	0.40	0.34
LightXML	0.32	0.21	0.26	0.29	0.28	0.29	0.30	0.29	0.30	0.28
ZestXML	0.56	0.45	0.50	0.63	0.60	0.61	0.67	0.65	0.66	0.59
CHRONOS	**0.75**	**0.61**	**0.67**	**0.80**	**0.75**	**0.77**	**0.82**	**0.79**	**0.80**	**0.75**
CHRONOS w/o DE	0.70	0.57	0.63	0.75	0.70	0.72	0.77	0.74	0.75	0.70
CHRONOS w/o TA	0.60	0.49	0.54	0.70	0.67	0.68	0.73	0.71	0.72	0.65

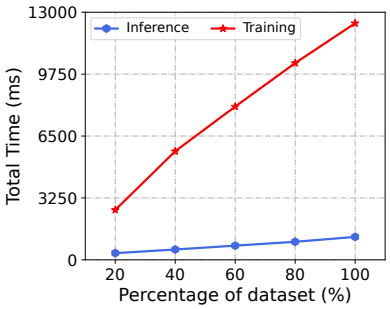


Fig. 3. Training and inference time of CHRONOS given different dataset sizes. The x-axis is the percentage of all vulnerability descriptions and the y-axis is the total time cost for training or inference. The total time of training or inference grows almost linearly with the size of dataset. It just costs just 1.62 milliseconds to train and 0.26 milliseconds for inference per vulnerability description.

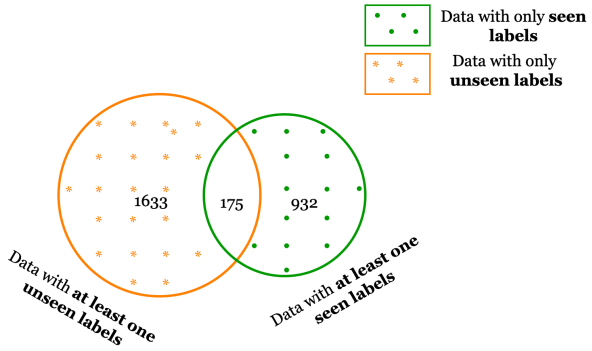


Fig. 4. Distribution of vulnerability reports. We have a total of 2,740 test instances, of which 1,633 are reports with unseen labels and 932 are reports with seen labels. 175 reports have both seen and unseen labels.

Answer to RQ3: All components of CHRONOS contribute positively to its effectiveness. Without data enhancement and time-aware adjustment, the performance of CHRONOS decreases by 6.7% and 9.2% in terms of average F1, respectively.

VI. DISCUSSION

A. Time Efficiency

For practical usage, CHRONOS should work under a reasonable amount of time. We investigate the efficiency of CHRONOS. We analyze the training time (the amount of time a model takes to learn all the training examples on average) and inference time (the amount of time a model takes to return all prediction results on average). The training time and inference time are related to two factors: the machine where the models run, and the size of the dataset (i.e., how many vulnerability descriptions are used to train or to infer). We limit the models to only using 8 CPU cores to simulate running on a regular consumer-grade laptop. As one would expect, a greater number of vulnerability descriptions takes a longer time to compute. There are 7,665 vulnerability descriptions in total. We experiment with different dataset sizes. To reduce the effects of randomness, we repeat the experiments three times. The results are presented in Figure 3.

The inference time is just a few thousands milliseconds and the training time is below fifteen thousand milliseconds with all vulnerability descriptions. The inference time and training time grow linearly with the size of the dataset, which shows the scalability of CHRONOS in practice. CHRONOS requires just 1.62 milliseconds for training and 0.26 milliseconds to infer labels for each vulnerability description. This indicates that CHRONOS is practical for use on a consumer-grade laptop.

B. Qualitative Analysis

As we reformulate the problem as a zero-shot learning task, we investigate the performance of CHRONOS in predicting labels without any training data. Figure 4 shows the composition of the vulnerability reports in the testing dataset.

We report the result of our approach, CHRONOS, on the data with only seen labels, some seen labels, and all data with respect to Precision, Recall, and F1 at top-k predictions (k=1,2,3). The detailed results are shown in Table VIII. CHRONOS achieves an average F1 of 0.70, with an F1@1 of 0.64, F1@2 of 0.72, and F1@3 of 0.75 on the data with **some** seen labels. For the data with **only** seen labels, our approach achieves an average F1 of 0.84, with an F1@1 of 0.75, F1@2 of 0.87, and F1@3 of 0.9. Comparing our CHRONOS's performance on data with only seen labels and data with some seen labels, CHRONOS performs better on the data with only seen labels by 20%.

TABLE VIII
COMPARISON OF CHRONOS UNDER DIFFERENT TESTING DATA. CHRONOS_FULLSEEN AND CHRONOS_PARTIALUNSEEN DENOTES THE RESULTS OF CHRONOS ON THE VULNERABILITY REPORTS WITH ONLY SEEN LABELS AND AT LEAST ONE UNSEEN LABELS, RESPECTIVELY

Model	P@1	R@1	F1@1	P@2	R@2	F1@2	P@3	R@3	F1@3	Avg. F1
CHRONOS_FullSeen	0.83	0.68	0.75	0.89	0.86	0.87	0.91	0.89	0.90	0.84
CHRONOS_PartialUnseen	0.70	0.58	0.64	0.76	0.69	0.72	0.77	0.73	0.75	0.70
CHRONOS	0.75	0.61	0.67	0.80	0.75	0.77	0.82	0.79	0.80	0.75

TABLE IX
COMPARISON OF THE EFFECTIVENESS OF CHRONOS WITH THE STATE-OF-THE-ART TECHNIQUES IN PREDICTING UNSEEN LABELS. CHRONOS IS ABLE TO CORRECTLY PREDICT 694 PREVIOUSLY UNSEEN LABELS.

Model	Success Rate	# Success Cases / Total
LightXML	0%	0 / 957
CHRONOS	**72.52%**	**694 / 957**

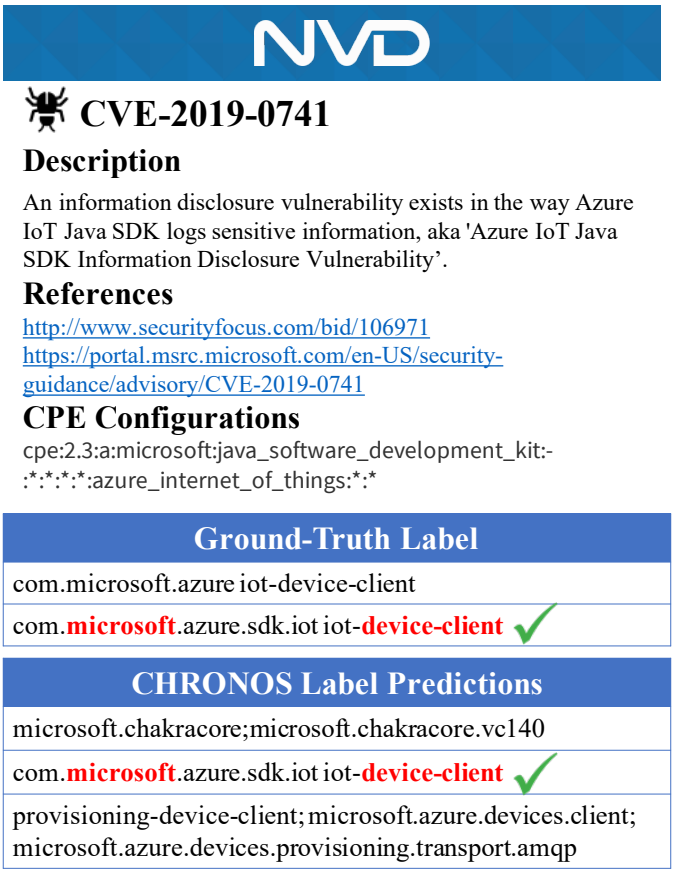


Fig. 5. Unseen Label Example. NVD entry for CVE-2019-0741. The top of the image shows its NVD entry with description, references, and CPE configurations. The bottom of the image shows the ground-truth label and CHRONOS's prediction.

This indicates that CHRONOS's performance on data with unseen labels still has room for improvement. Nevertheless, CHRONOS is able to perform reasonably well on the data with some seen labels. We conclude that CHRONOS achieves strong prediction results for the seen labels and is still effective at making good predictions on the unseen labels.

Table IX shows the improvements of CHRONOS over the baseline LightXML in predicting unseen labels. While LightXML cannot predict unseen labels, our method successfully predicts at least one correct vulnerability for 72.52% of them. Figure 5 shows an example of a vulnerability report where CHRONOS successfully predicts an unseen label. Given the text extracted from the vulnerability report, we compare the ground-truth label and the predictions of CHRONOS. The red, bold characters in the ground-truth label and CHRONOS's predictions are text that do not appear in the vulnerability description. Even if the library name does not explicitly appear in the vulnerability descriptions, our method can predict them successfully. This suggests that CHRONOS successfully learns to identify relevant terms that are indicative of each library.

C. Threats to Validity

Threats to internal validity include possible errors in our implementation. A possible threat relates to the selection of CHRONOS's and baseline approaches' hyper-parameters. To mitigate this threat, we tune these hyper-parameters using grid search, selecting the best parameters using the validation dataset. While we tuned LightXML using a grid search with fewer parameters compared to CHRONOS (learning rate, batch size, and epoch), LightXML's effectiveness is limited as it is fundamentally unable to predict previously unseen labels. We made the source code of our tool [13] and data [12] publicly available, allowing other researchers to validate our findings.

Threats to construct validity are related to the suitability of our evaluation metrics. To minimize this threat, we have used the same performance metrics of precision, recall, and F1 of the top 3 predictions that were used in the previous studies [6], [10]. These are standard metrics used in the literature of XML approaches [11], [14].

Threats to external validity are concerned with the generalizability of our experiments and findings. A possible threat is the dataset used in our experiments. We have utilized the same dataset from prior work [6], [10], containing vulnerabilities spanning over several years. These data were collected and validated by security researchers in Veracode, hence, we believe that the threat is minimal. A final threat is related to our experiments on LightXML. If we consider an experimental setting where the data is provided as a stream, LightXML can be retrained after seeing each data point. Unfortunately, LightXML requires 6 hours for training, which means that 6 (hours) x 2740 (entries) = 16,440 (hours) would be required for retraining LightXML. Therefore, a comparison of CHRONOS and LightXML when data is provided as a stream is very expensive. We emphasize that this is a limitation of

LightXML; LightXML will fail to predict previously unseen labels (occurring up to 70% in Table IX) without frequent retraining, while CHRONOS does not have this limitation.

VII. Related Work

Software Composition Analysis is increasingly essential for securing software systems. In recent years, there have been many studies investigating the dependencies of software [32]–[36]. Many empirical studies have reported the impact of vulnerable dependencies in the software supply chain. For example, Decan et al. [36] found that the number of vulnerability-affected packages in the npm network is growing over time, and half of the affected packages do not get fixed even when the fix is available. Lauinger et al. [37] also showed that around 37.8% of the packages in the `npm` network have at least one vulnerable dependency. These findings demonstrated the growing importance of securing the software supply chain.

Unfortunately, developers are slow in updating their vulnerable dependencies, leading to the risk of exploitation [8], [38]. Like our study, other researchers have recently proposed automated methods that have emerged as a promising solution for speeding up the process [39], [40]. For example, Mirhosseini et al. [39] found that projects that use automated pull requests upgrade dependencies 1.6x often as projects that did not use any tools. Other studies propose methods for detecting or obtaining more information about vulnerabilities from different software artifacts, including commits [41]–[45], bug reports [31], [46], [47], and mailing lists [48], [49]. Unlike these studies, we do not aim to detect vulnerabilities but to determine which libraries are vulnerable based on a vulnerability report. Other methods help developers to check if a library vulnerability can be exploited [50], [51] but already requires comprehensive information about the vulnerable library.

For vulnerability reports, researchers have proposed methods to assist in the analysis of vulnerabilities. Some studies focus on identifying affected versions [52] or predicting the exploitability of a vulnerability [53]. Other approaches use vulnerability reports to predict the key aspects, severity, or other properties of vulnerabilities [53]–[56]. Another method models new attack techniques from textual descriptions of vulnerabilities [57]. Similarly, our work aims to make predictions of vulnerability reports. However, we have a different goal of selecting libraries from a large space of possible labels.

Our study shows the importance of considering more practical experimental setups for analyzing vulnerability reports. Other Software Engineering studies have also shown that overlooking time and other practical concerns may lead to brittle experimental results [26], [58]–[62]. To address this practical challenge, our technique relies on a cache to leverage the time locality of the vulnerability reports. This phenomenon has been observed in other artifacts of software engineering. Tamrawi et al. [63] uses a caching strategy to enhance bug triaging by prioritizing developers who recently fixed related bugs. Caches of identifier names have been used to improve language models for source code [64]–[66].

VIII. Conclusion and Future Work

Software Composition Analysis (SCA) depends on significant human effort in identifying every library that is affected by a vulnerability report. Due to the large space of possible libraries, human effort can be error-prone. Manual analysis relies on the human annotator's limited domain knowledge to match libraries against vulnerability reports that may not explicitly indicate every relevant library. However, in this study, we show that the experimental setup considered in prior studies using extreme multi-label classification techniques may not consider a practical setting.

We reformulate the problem as a generalized zero-shot learning task, in which we face the challenge of predicting previously unseen labels. Under the more realistic setting, prior approaches face a substantial drop in performance.

CHRONOS uses zero-shot XML, data enhancement of the documents and labels, and time-aware adjustment of the labels. CHRONOS is frequently able to produce the right labels even if they were previously unseen. CHRONOS achieves an average F1 of 0.75, improving over the strongest approach identified in a prior study by 167.9%. Our experiments also indicate that each component of CHRONOS contributes to its effectiveness. These results suggest that the combination of techniques employed in CHRONOS successfully addresses the challenge of predicting previously unseen libraries. Overall, the experiments suggest that CHRONOS is effective for identifying libraries from vulnerability reports.

This study takes a large step forward in considering the real-world practical concerns of library identification from vulnerability reports. Among other techniques in CHRONOS, the use of reference data proved to be helpful in our experiments, however, the references listed on NVD entries may be incomplete as the references were also identified through human analysis. In the future, we will investigate methods of using software artifact traceability techniques [67], [68] to link the NVD report to related artifacts (e.g. the commits on GitHub fixing the vulnerabilities).

IX. Data Availability

CHRONOS's dataset and implementation are publicly available at https://figshare.com/articles/software/Chronos-ICSE23/22082075 and https://github.com/soarsmu/Chronos, respectively.

Acknowledgement

This project is supported by the National Research Foundation, Singapore and National University of Singapore through its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office under the Trustworthy Computing for Secure Smart Nation Grant (TCSSNG) award no. NSOE-TSS2020-02. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and National University of Singapore (including its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office).

Xuan-Bach D. Le is supported by the Australian Government through the Australian Research Council's Discovery Early Career Researcher Award, project number DE220101057.

REFERENCES

[1] G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo, "Out of sight, out of mind? how vulnerable dependencies affect open-source projects," *Empirical Software Engineering (EMSE)*, vol. 26, no. 4, pp. 1–34, 2021.

[2] N. Imtiaz, A. Khanom, and L. Williams, "Open or sneaky? fast or slow? light or heavy?: Investigating security releases of open source packages," *IEEE Transactions on Software Engineering (TSE)*, 2022.

[3] N. Imtiaz, S. Thorn, and L. Williams, "A comparative study of vulnerability reporting by software composition analysis tools," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–11.

[4] D. Foo, J. Yeo, H. Xiao, and A. Sharma, "The dynamics of Software Composition Analysis," *Automated Software Engineering (ASE) (Late Breaking Results)*, 2019.

[5] "Why do organizations trust Snyk to win the open source security battle?" https://snyk.io/blog/why-snyk-wins-open-source-security-battle/.

[6] Y. Chen, A. E. Santosa, A. Sharma, and D. Lo, "Automated identification of libraries from vulnerability data," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2020, pp. 90–99.

[7] "Snyk's database, CVE-2018-1914," https://security.snyk.io/vuln?search=CVE-2018-19149.

[8] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering (EMSE)*, vol. 23, no. 1, pp. 384–417, 2018.

[9] B. Chinthanet, R. G. Kula, S. McIntosh, T. Ishio, A. Ihara, and K. Matsumoto, "Lags in the release, adoption, and propagation of npm vulnerability fixes," *Empirical Software Engineering (EMSE)*, vol. 26, no. 3, pp. 1–28, 2021.

[10] S. A. Haryono, H. J. Kang, A. Sharma, A. Sharma, A. Santosa, A. M. Yi, and D. Lo, "Automated identification of libraries from vulnerability data: can we do better?" in *IEEE/ACM International Conference on Program Comprehension (ICPC)*, 2022.

[11] T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang, "LightXML: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 7987–7994.

[12] Y. Lyu, T. Le-Cong, H. J. Kang, R. Widyasari, Z. Zhao, X.-B. D. Le, M. Li, and D. Lo, "Dataset," https://figshare.com/articles/software/Chronos-ICSE23/22082075.

[13] ——, "Implementation," https://github.com/soarsmu/Chronos.

[14] Y. Prabhu and M. Varma, "FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 2014, pp. 263–272.

[15] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–37, 2019.

[16] N. Gupta, S. Bohra, Y. Prabhu, S. Purohit, and M. Varma, "Generalized zero-shot extreme multi-label learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 527–535.

[17] Z. Yang, J. Shi, S. Wang, and D. Lo, "Incbl: Incremental bug localization," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1223–1226.

[18] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020.

[19] A. Corazza, S. Di Martino, and V. Maggio, "Linsen: An efficient approach to split identifiers and expand abbreviations," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 233–242.

[20] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," in *2009 6th IEEE International Working Conference on Mining Software Repositories (MSR)*. IEEE, 2009, pp. 71–80.

[21] M. Hucka, "Spiral: splitters for identifiers in source code files," *Journal of Open Source Software*, vol. 3, no. 24, p. 653, 2018.

[22] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first Instructional Conference on Machine Learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.

[23] A. Buttner, T. Wittbold, and N. Ziring, "Common platform enumeration (cpe)-name format and description," NATIONAL SECURITY AGENCY/CENTRAL SECURITY SERVICE FORT MEADE MD FORT MEADE ..., Tech. Rep., 2007.

[24] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*, 2017, pp. 49–60.

[25] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep just-in-time defect prediction: how far are we?" in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2021, pp. 427–438.

[26] H. J. Kang, K. L. Aw, and D. Lo, "Detecting false alarms from automatic static analysis tools: How far are we?" in *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2022.

[27] M. I. Malinen and P. Fränti, "Balanced k-means for clustering," in *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, S+ SSPR 2014, Joensuu, Finland, August 20-22, 2014. Proceedings*. Springer, 2014, pp. 32–41.

[28] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[29] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[30] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNET: Generalized autoregressive pretraining for language understanding," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] Y. Chen, A. E. Santosa, A. M. Yi, A. Sharma, A. Sharma, and D. Lo, "A machine learning approach for vulnerability curation," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, 2020, pp. 32–42.

[32] J. Han, S. Deng, D. Lo, C. Zhi, J. Yin, and X. Xia, "An empirical study of the dependency networks of deep learning libraries," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 868–878.

[33] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering (EMSE)*, vol. 24, no. 1, pp. 381–416, 2019.

[34] S. E. Ponta, H. Plate, and A. Sabetta, "Detection, assessment and mitigation of vulnerabilities in open source dependencies," *Empirical Software Engineering (EMSE)*, vol. 25, no. 5, pp. 3175–3215, 2020.

[35] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 181–191.

[36] R. E. Zapata, R. G. Kula, B. Chinthanet, T. Ishio, K. Matsumoto, and A. Ihara, "Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm Javascript packages," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 559–563.

[37] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, "Thou shalt not depend on me: Analysing the use of outdated Javascript libraries on the web," *arXiv preprint arXiv:1811.00918*, 2018.

[38] A. Decan, T. Mens, A. Zerouali, and C. De Roover, "Back to the past–analysing backporting practices in package dependency networks," *IEEE Transactions on Software Engineering (TSE)*, 2021.

[39] S. Mirhosseini and C. Parnin, "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" in *2017 32nd IEEE/ACM international conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 84–94.

[40] B. Rombaut, F. R. Cogo, B. Adams, and A. E. Hassan, "There's no such thing as a free lunch: Lessons learned from exploring the overhead introduced by the greenkeeper dependency bot in npm," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2022.

[41] G. Nguyen-Truong, H. J. Kang, D. Lo, A. Sharma, A. E. Santosa, A. Sharma, and M. Y. Ang, "Hermes: Using commit-issue linking to detect vulnerability-fixing commits," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 51–62.

[42] J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, and A. E. Hassan, "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, 2021, pp. 705–716.

[43] T. G. Nguyen, T. Le-Cong, H. J. Kang, X.-B. D. Le, and D. Lo, "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*, 2022, pp. 1726–1730.

[44] Y. Zhou, J. K. Siow, C. Wang, S. Liu, and Y. Liu, "SPI: Automated identification of security patches via commits," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1–27, 2021.

[45] A. D. Sawadogo, T. F. Bissyandé, N. Moha, K. Allix, J. Klein, L. Li, and Y. Le Traon, "SSPCatcher: Learning to catch security patches," *Empirical Software Engineering (EMSE)*, vol. 27, no. 6, pp. 1–32, 2022.

[46] Y. Zhou and A. Sharma, "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, pp. 914–919.

[47] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, "How to better distinguish security bug reports (using dual hyperparameter optimization)," *Empirical Software Engineering (EMSE)*, vol. 26, no. 3, pp. 1–37, 2021.

[48] R. Ramsauer, L. Bulwahn, D. Lohmann, and W. Mauerer, "The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2020, pp. 147–157.

[49] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities," in *2006 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2006, pp. 6–pp.

[50] E. Iannone, D. Di Nucci, A. Sabetta, and A. De Lucia, "Toward automated exploit generation for known vulnerabilities in open-source libraries," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 396–400.

[51] H. J. Kang, T. G. Nguyen, B. Le, C. S. Păsăreanu, and D. Lo, "Test mimicry to assess the exploitability of library vulnerabilities," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2022, pp. 276–288.

[52] L. Bao, X. Xia, A. E. Hassan, and X. Yang, "V-SZZ: automatic identification of version ranges affected by CVE vulnerabilities," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022, pp. 2352–2364.

[53] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 105–114.

[54] H. Guo, S. Chen, Z. Xing, X. Li, Y. Bai, and J. Sun, "Detecting and augmenting missing key aspects in vulnerability descriptions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–27, 2022.

[55] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 125–136.

[56] X. Gong, Z. Xing, X. Li, Z. Feng, and Z. Han, "Joint prediction of multiple vulnerability characteristics through multi-task learning," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2019, pp. 31–40.

[57] H. Binyamini, R. Bitton, M. Inokuchi, T. Yagyu, Y. Elovici, and A. Shabtai, "An automated, end-to-end framework for modeling attacks from vulnerability descriptions," *arXiv preprint arXiv:2008.04377*, 2020.

[58] F. Tu, J. Zhu, Q. Zheng, and M. Zhou, "Be careful of when: an empirical study on time-related misuse of issue tracking data," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018, pp. 307–318.

[59] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 392–401.

[60] M. Christakis and C. Bird, "What developers want and need from program analysis: an empirical study," in *Proceedings of the 31st IEEE/ACM international conference on Automated Software Engineering (ASE)*, 2016, pp. 332–343.

[61] E. B. Sørensen, E. K. Karlsen, and J. Li, "What norwegian developers want and need from security-directed program analysis tools: A survey," in *Proceedings of the Evaluation and Assessment in Software Engineering*, 2020, pp. 505–511.

[62] E. Winter, D. Bowes, S. Counsell, T. Hall, S. Haraldsson, V. Nowack, and J. Woodward, "How do developers really feel about bug fixing? directions for automatic program repair," *IEEE Transactions on Software Engineering*, 2022.

[63] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (FSE)*, 2011, pp. 365–375.

[64] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*, 2017, pp. 763–773.

[65] C. Franks, Z. Tu, P. Devanbu, and V. Hellendoorn, "CACHECA: A cache language model based code suggestion tool," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, vol. 2. IEEE, 2015, pp. 705–708.

[66] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014, pp. 269–280.

[67] A. D. Rodriguez, J. Cleland-Huang, and D. Falessi, "Leveraging intermediate artifacts to improve automated trace link retrieval," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 81–92.

[68] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 324–335.

TABLE X
COMPARISON OF THE EFFECTIVENESS OF CHRONOS WITH THE STATE-OF-THE-ART TECHNIQUES. THE BOLD NUMBERS DENOTE THE BEST RESULTS FOR EACH METRIC.CHRONOS W/O DE, CHRONOS W/O TA DENOTES THE RESULTS OF CHRONOS WITHOUT DATA ENHANCEMENT AND TIME-AWARE ADJUSTMENT, RESPECTIVELY. WE USE ORIGINAL P@K IN THIS TABLE.

Model	**P@1**	**R@1**	**F1@1**	**P@2**	**R@2**	**F1@2**	**P@3**	**R@3**	**F1@3**	**Avg. F1**
Exact Matching	0.33	0.26	0.29	0.27	0.41	0.32	0.20	0.46	0.28	0.30
CPE Matcher	0.27	0.26	0.26	-	-	-	-	-	-	-
Traditional IR	0.34	0.25	0.29	0.23	0.33	0.27	0.19	0.39	0.25	0.27
LightXML	0.32	0.21	0.26	0.24	0.28	0.26	0.18	0.29	0.22	0.25
ZestXML	0.56	0.45	0.50	0.39	0.60	0.47	0.29	0.65	0.40	0.46
CHRONOS	**0.75**	**0.61**	**0.67**	**0.49**	**0.75**	**0.60**	**0.36**	**0.79**	**0.49**	**0.59**
CHRONOS w/o DE	0.70	0.57	0.63	0.46	0.70	0.56	0.36	0.74	0.46	0.55
CHRONOS w/o TA	0.60	0.49	0.54	0.43	0.67	0.52	0.32	0.71	0.44	0.50

APPENDIX

A. Evaluation Results using the Original Precision@K

In this paper, we use a normalized version of $P@k$ to avoid the denominator being larger than the number of actual labels. In this appendix, we present the evaluation results on the original $P@k$, which is calculated as follows:

$$P@k(v) = \frac{lb\_k(v) \cap \hat{lb}(v)}{k}$$

where $lb\_k(v)$ and $\hat{lb}(v)$ are top-k prediction and the actual labels for a given vulnerability report v.

Table X illustrates that our primary conclusions and contributions remain consistent under the original $P@k$ setting. Specifically, CHRONOS outperforms the standard zero-shot learning model in terms of Avg. F1 by 28%, and it outperforms LightXML by 136%.