AVMiner: Expansible and Semantic-Preserving Anti-Virus Labels Mining Method

Ligeng Chen[†§]✉, Zhongling He[†], Hao Wu[†], Yuhang Gong[†] and Bing Mao[†]
[†]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
{chenlg, zhe, gyh}@smail.nju.edu.cn, {hao.wu, maobing}@nju.edu.cn
[§]Asiainfo Security Technologies Co.,ltd

***Abstract*—With the increase in the variety and quantity of malware, there is an urgent need to speed up the diagnosis and the analysis of malware. Extracting the malware family-related tokens from AV (Anti-Virus) labels, provided by online anti-virus engines, paves the way for pre-diagnosing the malware. Automatically extract the vital information from AV labels will greatly enhance the detection ability of security enterprises and equip the research ability of security analysts. Recent works like AVCLASS and AVCLASS2 try to extract the attributes of malware from AV labels and establish the taxonomy based on expert knowledge. However, due to the uncertain trend of complicated malicious behaviors, the system needs the following abilities to face the challenge: preserving vital semantics, being expansible, and free from expert knowledge. In this work, we present AVMiner, an expansible malware tagging system that can mine the most vital tokens from AV labels. AVMiner adopts natural language processing techniques and clustering methods to generate a sequence of tokens without expert knowledge ranked by importance. AVMiner can self-update when new samples come. Finally, we evaluate AVMiner on over 8,000 samples from well-known datasets with manually labeled ground truth, which outperforms previous works.**

***Index Terms*—AV Labels, Expansible, Natural Language Processing**

I. Introduction

According to the report of AV-TEST[1], 130 million new viruses were detected in the past year (4.1 viruses per second on average). Rapid mutating malware has a great impact on daily lives, causing huge losses to people's properties. Hence, precisely tagging a suspicious malware in advance will significantly help security enterprises to archive the malware by different families, and assist security analysts in further analysis, even patching the vulnerabilities.

Some online services like VirusTotal[2] provide malware labels detected by various anti-virus engines, which are called AV labels. The naming way of AV labels follows the original intention of MAEC (Malware Attribute Enumeration and Characterization)[3], aiming to propose a structured representation with high-fidelity information about the attributes of malware. However, since the labels are produced by different vendors independently, they are always inconsistent with each other on the judgment of characteristics (i.e., malicious or benign), descriptions (e.g., malware class, property, and behavior), and so on. Previous works have located the label inconsistency problem. What's more, AV-Meter[4] empirically studies the correctness and inconsistency of malware tagging on manually labeled datasets. Even well-known vendors can not always perform well. Based on the observation that tags change over time, Zhu et al.[5] thoroughly research and show that AV labels would flip over time, speculating AV vendors produce strongly correlated labels.

Even though the massive AV labels are noisy, some works[6], [7], [8] demonstrate that the ground-truth of tags is hidden in the given labels, which can be automatically extracted with rules built by expert knowledge.

Unfortunately, due to the unpredictable trend of malware, the complicated relationships between malicious behaviors, and the inconsistent description of AV labels, we are still facing the following challenges of automatically extracting the correct labels for the malware samples. ❶ Complex relationships of malicious behaviors and vendor naming rules are challenging to be exhausted by the limited rules. Is it possible to extract the labels and reunite the relationships between them ***Without Expert Knowledge***? ❷ Although different AV vendors produce inconsistent labels, the labels show an inherent relation. The different detection mechanisms (e.g., static analysis, sandbox execution), focus on different aspects of the malware. A single label may only describe a malicious behavior profile, but breaking up and regrouping the massive AV labels may give a full picture of the sample. So how to ***Preserve Vital Semantics*** rather than directly follow the principle of the minority obeying the majority? ❸ The arms race between hackers and security researchers is increasingly fierce. Researchers are unable to predict the characteristics of the new kind of viruses before their appearance, let alone categorize them. Some new malware, e.g., HeartBleed[9], EternalBlue[10], and Vjworm[11], cannot be classified into any malware type. So how to make the system ***Expansible*** adapt to the development of malware?

To deal with the problems mentioned above, we present a novel malware tagging method called AVMiner (**A**nti-**V**irus **Miner**), which can adapt to new data with no human effort. ① AVMiner adopts state-of-the-art NLP (Natural Language Processing) techniques and adaptive clustering techniques to extract the labels for the malware, rather than establish excessive rules with massive expert knowledge. The system will tokenize, generalize and extract the key label(s) and orchestrate them according to their relations ***Without Expert Knowledge***. ② By utilizing machine learning techniques, we can be freed from extracting endless rules. We try to ***Preserving the Vital***

Semantics as much as possible, based on the principle of co-occurrence frequency. The extracted labels will be reunited to a relation graph, which will give a full picture of the malware to the users. ③ Due to the system design, AVMiner requires almost no expert knowledge. With the mutation of malware, it is able to update self-adaptively to deal with coming malicious samples and its AV labels. So the ***Expansibility*** of the system can meet the community's needs.

We evaluate AVMiner on over 8,000 samples with manually labeled ground truth and compare it with state-of-the-art malware label extraction method. AVMiner achieves an accuracy of 93.5% when outputs 3 labels, and 97.9% when outputs 5 labels, separately higher than AVCLASS and AVCLASS2 by about 7% and 9%. Also, we evaluate AVMiner on about 2 million samples without ground truth, which can give the output as long as the AV labels are given. But AVCLASS and AVCLASS2 fail to output the result for 21.8% and 10.6% of samples. AVMiner shows great robustness and expansibility on the diverse distribution of malicious samples.

The main contributions of our work are as follows:

1) We propose a novel malware label generation method by mining the malware label from the massive AV labels collected from security communities to accelerate the malware diagnosing procedure.
2) We design a new tool AVMiner to effectively generate the tokens that express malware's security semantics from the multi-source and noisy AV labels. AVMiner works without expert knowledge and can self-adaptively update with the newly come samples.
3) We evaluate the AVMiner on 2 manually labeled datasets and measure it on about 2 million malicious samples without ground truth. Experimental results show that AVMiner behaves better than the existing works.

II. System Design

A. Overview

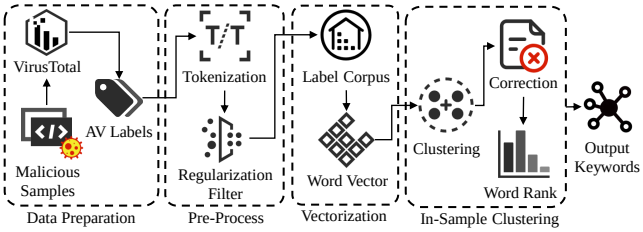

Fig. 1: Workflow of AVMiner.

In this section, we provide an informal overview of our method AVMiner on an illustrative example as shown in Figure 2. According to Figure 1, AVMiner mainly contains five parts: *data preparation*, *pre-processing*, *vectorization*, *in-sample clustering* and *keywords output*, each of which will be introduced in detail in the following subsections.

B. Data Preparation

The Data Preparation phase is shown on the left side of Figure 1. In this phase, we first collect the malware' detected results from the VirusTotal (denoted as AV label). Each AV label contains a sequence of keywords indicating the detected result from each vendor assembled on VirusTotal, as shown in Figure 2-1. These AV labels describe the possible behavior, attributes, and categories of a malicious sample (e.g., "Win32Flystudio.worm.Gen" produced by *AhnLab-V3*, "Win32" denotes the platform, "Flystudio" denotes the category, "worm" denotes the behavior). In the following phases, AVMiner will process the AV labels into an intermediate representation and extract the keywords best describing the samples (i.e., "Flystudio" in the aforementioned example) for the users.

To free from summarizing excessive rules and make the best use of advanced techniques, our AVMiner is based on statistical results and equipped with machine learning methods. To deal with the problem of cold start with zero knowledge, we gather a considerable amount (more than 8,000 in the evaluation) of samples to set up the corpus in the following phase. When new samples come, they can be directly added to the system and make the tool adaptive to unknown kinds of malware. Abundant and diverse datasets of malicious samples can not only make our systems more inclusive for different kinds of malware but also enable us to build a virus-related knowledge graph in the future.

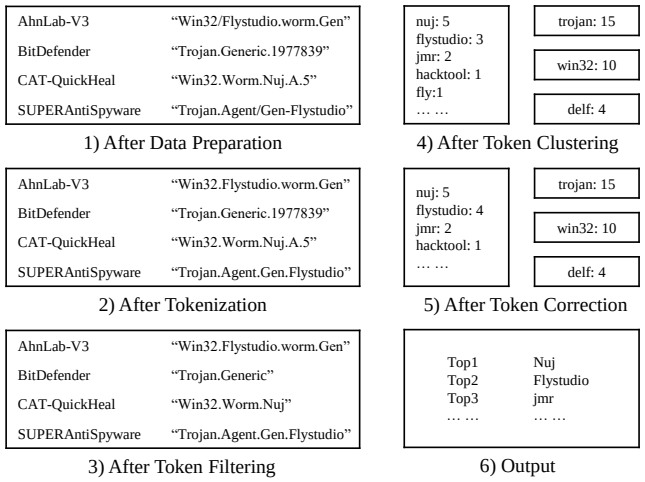

Fig. 2: Running example. The left side of steps 1) to 3) denotes the name of AV vendors, right side denotes the AV labels and processed tokens. Step 4) and 5) contains different clustered tokens and their amount. Step 6) outputs the ranked tokens.

C. Pre-Processing

The pre-processing phase consists of two parts: tokenization and token filter. The tokenization aims to split the AV labels into tokens for the same token merging and related tokens clustering. The token filtering aims to filter out the tokens with a few semantics and prevent them from blocking the subsequent stages.

a) Tokenization: The AV labels are sequences of tokens and are highly vendor-dependent. For example, in the face of a Trojan sample which can be classified as *delf*, *Emsisoft* describes it as *Trojan-Dropper.Delf!IK*, but it is described as *Trojan.DL.Delf!qqcViDnxCRM* by *VirusBuster*. The various strategies of AV vendors to tag the malware leads to the AV

labels appearing in quite different formats but with even the same semantics. So it is vital to split the AV labels to find the associated parts between the inconsistent labels. What's more, the trend of malware development is unpredictable, and AV vendors may update tagging strategies over time. But it takes time to customize rules that cause the rules to always be out-of-date. So customizing a set of constant rules is hard. To address the challenges mentioned above, we first tokenize the AV labels into the smallest units, called tokens, which are separated by some specific punctuations in AV labels. Specifically, the compound words in our scenario should be preserved, because these words usually have specific semantics in a particular context. To clearly present the result, we replace all the punctuations with a unified separator, denoting the tokenization operation. According to the example in Figure 2-2, the AV label *Win32/Flystudio.worm.Gen* produced by the vendor *AhnLab-V3* is related to a malware sample. We process the AV label into *Win32.Flystudio.worm.Gen*, containing 4 tokens, separated by a dot, leaving alone the compound word *Flystudio* and the abbreviation *Gen* (representing for word *generic*).

b) Token filter: Performing tokenization may produce a large number of redundant and meaningless tokens, so we need to remove them, aiming to minimize the influence of noise. For example, according to Figure 2-2), AV label "Win32.Worm.Nuj.A.5" produced by *CAT-QuickHeal* comprises two tokens *A* and *5*, which seem to have low-security semantics for the malware and may only be used as a malware identifier or only as a serial number. Although we find this kind of token often appears in some fixed locations, it cannot be removed directly due to the following reason. The token number of each AV label is quite different, ranging from 1 to 6. Even the AV labels produced from the same vendor may not be aligned. For example, the token number of AV labels produced by *BitDefender* is ranging from 3 to 5.

Through a detailed investigation of the collected AV labels, we find them there are some key differences between the meaningful tokens and the meaningless ones. *First*, the AV labels from the same vendor have almost the same formal structures. For example, the first token of *BitDefender* is usually a feature description (e.g., generic, trojan), the second is usually a file type description (e.g., JS), and the last token is usually a number or identifier. Although the length may vary depending on the description, the same type of tokens is generally in the same relative position. *Second*, the meaningful tokens usually have a high repetition frequency, and they are often used to describe an attribute, family, class, etc, while the meaningless tokens are only used as an identifier with a low repetition frequency. So the token position and the token distribution light the way for us to effectively filter the meaningless tokens. We come up with a strategy called ***Unique Tokens Fade Away***. According to the observation, the same relative token position (in order or reverse) of AV labels produced by the vendor always contains the same type of tokens as mentioned. And the meaningless tokens are unique in the same relative position. When we count the proportion of unique tokens in all the tokens, this may lead to a clear distinction. Here we define the unique index σ as,

$$\sigma(i, i_{max}, v_t, f) = \begin{cases} \frac{T_u(i,v_t)}{T_a(i,v_t)}, f = 1 \\ \frac{T_u(i_{max}-i,v_t)}{T_a(i_{max}-i,v_t)}, f = 0 \end{cases} \quad (1)$$

where σ indicates the proportion of unique tokens (i.e., tokens appear only once) in all the tokens. The calculation of σ is restricted to the same relative position of the same vendor. The parameters of σ are defined as follows: i denotes the i^{th} position of the AV label, i_{max} denotes the longest token sequence in the same vendor, v_t denotes the t^{th} AV vendor, f denotes a flag whether we count the token orderly (f=1) or reversely (f=0). $T_u(i, v_t)$ denotes the amount of the unique tokens in the i^{th} position from the t^{th} AV vendor, $T_a(i, v_t)$ denotes the amount of all the tokens in the i^{th} position from the t^{th} AV vendor.

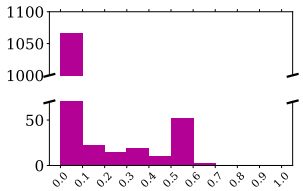

Fig. 3: Distribution of index σ.

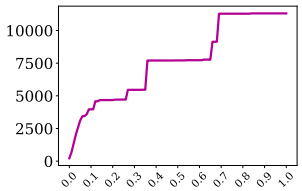

Fig. 4: Remain amount.

To determine the specific value of the unique index σ, we empirically investigate the relationships between different values of σ and their result. The distribution of σ is shown in Figure 3, the horizontal axis of which denotes the range of σ, and the vertical axis of which denotes the number of different positions of vendors corresponding to the range of σ. According to the figure, most of the positions have a few unique tokens (i.e., σ<0.1). When σ is around 0.5, there is a peak, which we think may be the position where a large number of meaningless tokens appear. Moreover, we count the remaining tokens when we adopt a different value of σ, which is shown in Figure 4. We find that there is a spike of around 0.3 and 0.7. We speculate that this may be due to the sudden introduction of a large number of meaningless tokens when the values are greater than 0.3 and 0.7, resulting in this phenomenon. Based on the above observations and analysis, we set the σ at 0.3. For each vendor, we separately process the tokens column by column of the corresponding AV labels until σ drops below 0.3. We keep scanning the AV labels orderly and reversely, to *fade away* the meaningless tokens, processed result of which is shown in Figure 2-3.

D. Vectorization

By a thorough investigation, we find GloVe (**Glo**bal **Ve**ctors)[12] is a suitable vectorization method in our scenario. GloVe is good at generalizing the co-occurrence relationship in the *global* scope. What's more, according to the algorithm design, GloVe does not use neural networks, and the computational complexity is independent of the data size, which is conducive to system expansion. The main procedures of this technique are first to count the co-occurrence matrix for each token with a fixed counting window and then to

reduce the dimension of the sparse matrix while ensuring the co-occurrence relation between each word. The target function is designed to make related tokens closer to the higher dimension, while irrelated tokens further, shown as follows,

$$J = \sum_{i,j=1}^{N} f\left(X_{ij}\right)\left(T_i^T \tilde{T}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \quad (2)$$

where T_i^T denotes the transpose of i^{th} token vector, T_j denotes the vector of j^{th} token, b_i and $\tilde{b}_j$ denote the bias terms, X_{ij} denotes the amount of j^{th} token appearing in the window of i^{th} token, function f denotes the weight function, and N is the size of the token vocabulary.

E. In-Sample Clustering

In this stage, we regroup the tokens by samples. After being processed by the former steps, we transform the tokens into vectorized representations, trying to map the context-aware relationships to the higher dimensions. AVMiner aims to output the most related token or token set to the malicious sample. So if the related tokens and the irrelated tokens can be classified into two groups, the output can be produced much more effectively. Fortunately, according to our observation, related tokens are always appearing together (e.g., *Trojan*, *Redirctor* and *Hidelink*), the transformed vectors of which are also closer in the higher dimension. Due to the situation of lacking ground truth (for supervised machine learning), a suitable unsupervised clustering technique can pave the way for the final output.

a) Token Clustering: There are several clustering algorithms that adopt different strategies to fit diverse situations. To meet our needs, we have some requirements for the clustering method. *First*, the lower the computational complexity, the better. As the system iterates, more new samples are introduced, and the calculation speed greatly affects the performance of the system. *Second*, the algorithm should be free from presetting the number of clusters. Because we can not predict how many clusters will be produced from one sample.

According to the aforementioned analysis result and some empirical experiments, we leverage Mean Shift[13] as our clustering algorithm. Mean Shift is a non-parametric feature-space analysis technique, which is always applied in cluster analysis. In other words, the algorithm calculates the offset means of the current point by specifying the radius of a high-dimensional sphere (i.e., parameter *bandwith*) and by continuous iteration, so that it can move to the space containing the most points at the same time. This characteristic quite fits our requirements, and we also hope to find the set of tokens with the highest correlation, which may contain the most accurate descriptive tokens of the malicious samples, as shown in Figure 2-4.

b) Token Correction: After grouping the tokens according to their relevance, we can sort them according to their frequency and the attributes of the clusters. However, we find that there is no uniform naming convention among the AV vendors, so some of the tokens may imply the same word, but with several different letters, even because vendors come from different countries. There are several reasons why we are considering token correction in the current step. On the one hand, because the computational complexity of pair comparison is $O(n^2)$, it takes a long time to make a comparison in the whole corpus. On the other hand, there may be mis-correction in the algorithm design, and we hope to minimize the side effects.

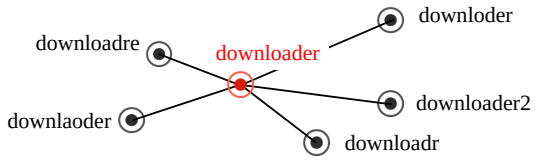

Fig. 5: An Example of the Correct Token and Mistaken Tokens.

As shown in Figure 5, it is a real case from our token corpus. All the tokens imply the same word *downloader*, but only the token colored with red (downloader) is spelt correctly. Because the frequency of the token plays an important role in the final rank, we may lose the weight of key tokens by mis-spelling. For instance, a malicious sample output the regrouped tokens (in one cluster) as shown in Table I, ranked by token frequency. Row 1 and 2 denote the tokens and their amount before being corrected, and rows 3 and 4 denote the ones after the procedure. The manually labeled ground truth of this sample is *plankton*, but several vendors mis-spell the words (i.e., plangton), so the key token *plankton* loses some frequency weight, which may lead to the wrong output. Once we correct the words and refresh the rank by frequency (from rank 5 to rank 2), we may have a higher probability of outputting the right words in the final stage.

TABLE I: Example of token correction.

Before Correction		After Correction	
Token	#	Token	#
andriod	13	andriod	13
trojan	11	**plankton**	12
airpush	8	trojan	11
plangton	6	airpush	8
plankton	6	adware	4

To complete token correction without affecting normal tokens, two prerequisites are given. Firstly, the correction task is only limited to the same cluster within the sample, minimizing the computational complexity (the complexity of pair comparison is $O(n^2)$). Secondly, We raise an error correction threshold δ to detect the tokens that need to be corrected, and the token will be corrected only when it reaches this threshold, which is defined as follows,

$$\delta = \frac{\mathsf{Edit}(\mathsf{T}_i, \mathsf{T}_j) - \mathsf{LenDiff}(\mathsf{T}_i, \mathsf{T}_j)}{\mathsf{MaxLen}(\mathsf{T}_i, \mathsf{T}_j)} \quad (3)$$

where $\mathsf{Edit}(\cdot,\cdot)$ denotes the Edit Distance (minimum number of edits for string conversion) between 2 tokens (i.e., T_i and T_j), $\mathsf{LenDiff}(\cdot,\cdot)$ denotes the length difference, and $\mathsf{MaxLen}(\cdot,\cdot)$ denotes the max length among them. The reason for setting the threshold in this way is that, not only can it correct a few different token pairs (e.g., *downloader* and

downloadre), but also it can also find the abbreviations of token pairs (e.g., *gen* and *generic*). When the threshold δ is below 0.3, we think there are candidates in the token pair that need to be corrected, and if a *standard word* (according to Wiki) does not exist in them until a *standard word* associated with them is found. The result is shown in Figure 2-5.

F. Keywords Output

To enhance the readability of the output, we try to rerank the tokens, which also benefits from customizing the amount of output. Besides the clustering relations and token frequency, which can be leveraged as ranking criteria, more global information should also be taken into consideration. Thanks to the technique TF-IDF (Term Frequency–Inverse Document Frequency)[14], which is a common weighting technique for information retrieval and data mining. Which will be ranked higher. The method is calculated as,

$$TF_{T_i,L(mal_j)} = \frac{\text{count}(T_i)}{|L(mal_j)|} \tag{4}$$

$TF_{T_i,L(mal_j)}$ (i.e., abbreviation of ***T**erm **F**requency*) denotes the frequency of token T_i in the malicious sample $L(mal_j)$, $count(T_i)$ denotes the frequency amount of token T_i, and $|L(mal_j)|$ denotes the amount of all the tokens in the malicious sample $L(mal_j)$.

$$IDF_{T_i} = \log \frac{N}{\sum_{j=1}^{N} I\left(T_i, L(mal_j)\right)} \tag{5}$$

IDF_{T_i} (i.e., abbreviation of ***I**nverse **D**ocument **F**requency*) reflects the universality of token T_i in the corpus, N denotes the amount of malware, $I\left(T_i, L(mal_j)\right)$ denotes whether the malware $L(mal_j)$ contains the token T_i (1 for positive and 0 for negative). However, if the token T_i does not appear in all the malware (e.g., when the dataset changes), then the denominator in the formula equals 0. So we need to smooth it, which is shown below,

$$IDF_{T_i} = \log \frac{N}{1 + \sum_{j=1}^{N} I\left(T_i, L(mal_j)\right)} \tag{6}$$

with the equations above on hold, we can calculate the $TF-IDF$ value as follows,

$$TF-IDF_{T_i,L(mal_j)} = TF_{T_i,L(mal_j)} * IDF_{T_i} \tag{7}$$

According to the definition and formulas, when the frequency of a token in the malicious sample is higher, and the rarity is higher, its TF-IDF value is higher.

By utilizing the clustering relations, token frequency, and TF-IDF value, the tokens can be reranked in each malicious sample by several steps, which are shown in the following pseudocode. Firstly, we need to find the *best cluster* as the candidate cluster, which contains the token with the highest TF-IDF value. According to our observation, the *best cluster* always comprises the tokens highly related to each other and to the malicious samples. Next, by judging whether the amount of tokens in the best cluster is sufficient, we will treat different situations separately. If the amount is sufficient, the first N tokens in the best cluster are added to the final result. And if the token has the second-highest TF-IDF value, which also indicates a relatively high correlation, we will replace it with the N^{th} token in the result. Otherwise, if the amount is insufficient, all the tokens in the best cluster are added to the result, the rest of which will be filled by the tokens according to the TF-IDF value. Finally, the result is presented to users in reranked tokens as shown in Figure 2-6.

Algorithm 1: Token rerank and output

```
Input: Clusters: Token clusters, as a 2D array. Each array,
       Clusters[ClusterID] denotes a token cluster, as an array of tokens,
       where the rank of the tokens indicates the token frequency.
       TFIDF: Array of tokens, ranked by TFIDF
       TopN: Amount of output tokens
Output: Result: Array of tokens, sorted by importance
Initialize: Result = ∅
/* Find the ID of the Best Cluster                                  */
for ClusterID from 1 to Clusters.Length do
    if TFIDF[1] in Clusters[ClusterID] then
        BestClusterID ← ClusterID
    end
end
/* The Amount of Tokens in the Best Cluster is
   Sufficient                                                       */
if Clusters[BestClusterID].Length ≥ TopN then
    Result ← Clusters[BestClusterID][1:Top(N)]
end
if TFIDF[2] not in Clusters[BestClusterID] then
    Result[TopN] ← TFIDF[2]
end
/* The Number of Tokens in the Best Cluster is
   Insufficient                                                     */
Result ← Clusters[BestClusterID]
for Index=1; Result.Length < TopN
  and Index < TFIDF.Length; Index+=1 do
    if TFIDF[Index] not in Result then
        Result ← Result+TFIDF[Index]
    end
end
```

III. Evaluation

In this section, we first describe the implementation in detail. Then, the dataset we use to evaluate AVMiner is introduced, followed by the comparison of tagging accuracy with prior works AVclass and AVclass2. Finally, we measure AVMiner's robustness in terms of dataset size, file type, and sample detected time.

A. Implementation

In this section, we present the implementation of our system, including the specific experimental details and parameters.

Firstly we retrieve data from VirusTotal. Once the retrieval finishes, we analyze all the data to filter the tokens, removing potentially meaningless tokens, using σ=0.3. Afterward, we train a GloVe model, where the window size is 40, the vector length is 32, and the training epoch is 100. Then, we use Mean Shift, where bandwidth is 2 and training epoch is 100, and TF-IDF to cluster tokens inside each malicious sample. Finally, we correct tokens inside each cluster with δ=0.3.

The Python implementation uses the following libraries: *glove*[15] and *gensim*[16] for vectorization, *scikit-learn*[17] for

TF-IDF, *editdistance*[18] for words correction, *tabulate*[19] for visualization. The evaluation part uses GNU Parallel[20] to speed up.

All our experiments are conducted on a PC with 16 GB memory, and 1 Intel i7-7700k CPU (4.2 GHz).

B. Dataset

We evaluate AVMiner on the datasets shown in Table II. Drebin[21] and Malheur[22] are manually collected and labeled malware datasets, which are also used in prior works. These two datasets are well-known and also widely used by several related works. Please note that our AVMiner is free from domain knowledge, and it can adapt to newly come malicious samples with fine-tuned models.

We also get a Superset by combining the two datasets. Column 2 to 5 separately denotes the platform, the number of malicious samples, the time range of sample collection, and whether the dataset has manually labeled ground truth.

TABLE II: Datasets used in the evaluation.

Dataset	# of Virus Type	Amount	Time Range	with G.T.
Malheur[22]	24	3,086	08/2009 - 08/2009	✓
Drebin[21]	178	5,511	08/2010 - 10/2012	✓
Superset	202	8,597	08/2009 - 10/2012	✓

We present the statistical results on file type as follows to show the dataset more clearly.

TABLE III: Malheur (Type)

File Format	Amount
Win32 EXE	2945
DOS EXE	139
unknow	2

TABLE IV: Drebin (Type)

File Type	Amount
Android	5503
others (ELF, GZIP, ZIP)	3
unknown	5

Table III and IV present the distribution of file formats (i.e., a standard way that information is encoded for storage in a computer file) in our datasets. Malheur mainly consists of windows executables (95.4%), with several DOS executables (4.5%) and 2 samples that cannot be classified by VirusTotal. Meanwhile, Android accounts for 99% of dataset Drebin, with a few ELF, GZIP, ZIP, and 5 undetectable samples. They contain malicious samples from 2 mainstream mobile and PC platforms, based on which we can better demonstrate that AVMiner is available for the commonly used platforms.

C. Overall Accuracy

Our experiments do not introduce external data without clarity to augment the dataset described in Section III-B, which can show that our AVMiner is not dependent on a large amount of available data and that even a small amount of incremental data can yield substantial results.

Table V presents the evaluation result on dataset Malheur and Drebin, by comparing AVMiner with prior works AVCLASS and AVCLASS2. Row 3 to 12 denote the tagging accuracy with $TopN$ ranging from 1 to 10 (i.e., if the manually labeled ground truth is the same as either token in the result of the concatenation of them, we take it as the positive case). Since AVCLASS produces only one token for a single malicious sample, its accuracy will remain the same no matter how $TopN$ changes. When we choose Top1, AVCLASS performs better on both datasets than others, because it gives a very clear and condensed classification result for a malicious sample, with the help of rich expert knowledge. Once we choose Top2, AVMiner goes beyond other tools. As we expand the chosen range, the tagging accuracy of AVMiner continues to rise. The reason that AVMiner cannot precisely hit the ground truth within Top1 is that AVMiner sometimes place some generic tokens at first, such as *Android*, *JS*, etc. In our system design, we do not remove the tokens describing the file format, which is a benefit for users to more comprehensively understand the malicious samples. So the tokens describing categories of behaviors may be placed to the back. But when the chosen scope is expanded, they come into the output token list.

TABLE V: Tagging Accuracy of Malheur and Drebin.

TopN	Malheur			Drebin		
	AVMiner	AVCLASS	AVCLASS2	AVMiner	AVCLASS	AVCLASS2
1	0.744	0.811	0.376	0.866	0.899	0.039
2	0.859	0.811	0.558	0.914	0.899	0.897
3	0.896	0.811	0.570	0.973	0.899	0.933
4	0.919	0.811	0.721	0.979	0.899	0.940
5	0.926	0.811	0.782	0.980	0.899	0.942
6	0.946	0.811	0.799	0.981	0.899	0.942
7	0.947	0.811	0.799	0.982	0.899	0.942
8	0.948	0.811	0.800	0.983	0.899	0.942
9	0.948	0.811	0.802	0.985	0.899	0.942
10	0.950	0.811	0.830	0.985	0.899	0.942

To further explore the different performances (accuracy and trend) on the different datasets, we try to deeply analyze the result and find the root cause. Intuitively, we take the diversity of malware types, the amount of the dataset, and the time range of collecting the samples. We first investigate the tagging accuracy of different virus types of ground truth. In Malheur, there are 24 types of virus, which are relatively evenly distributed, with *allaple*, *podnuha*, and *rotator* having the most samples, all with 300. While in Drebin, the taxonomy is more detailed. There are 178 virus types in all, type *fakeinstall* has the most amount of samples (907), and some virus types have only one sample (e.g., *acnetdoor*, *cellshark*).

We investigate the accuracy of each virus type. Even Drebin contains a more diverse virus type of malicious samples, the majority of the dataset is made up of a few kinds of virus types, e.g., *Fakeinstaller*, *DroidKungfu*, and *Plankton*. Most of the AV vendors can effectively label these categories, so it paves the way for the extraction. However, sometimes most of the AV vendors cannot label the malicious samples, so it blocks us from label extraction. As for Malheur, one of the majority labels, Spygames, which takes up 4.5%, fails to be distinguished by almost all of the AV vendors, only except for AntiVir. What's worse, the output of AntiVir for this type of malicious sample is "TR/Spy.Games.A", causing all 3 tools' tokenization strategy to split it as "Spy", "Games", and "Spy" and "Games" do not appear in the *Best Cluster*, which leads to the failure. As for AVCLASS and AVCLASS2, they rank the labels only on their frequencies, causing ignorance. On the other hand, we could extract some categories (e.g., *rbot*, *zhelatin*) with 100% accuracy but AVCLASS and AVCLASS2 miss all of the samples. We speculate that the expert knowledge-based method sometimes leads to failure since it cannot cover all the situations. But we choose to benefit from all the vendors, which equips AVMiner with more expansibility.

Since AVMiner employs unsupervised machine learning, we

suspect that the performance may rely on the distribution and amount of malicious samples. We apply the same experiment to the Superset, and the result is shown in Table VI. The result shows that the performance on supersets is even better than the weighted average of the subsets. Parts of the datasets may gain from each other (i.e., context relations), improving the performance of previously underperforming parts. If so, it indicates that the more related samples introduced, the better the performance will be.

TABLE VI: Tagging Accuracy Comparison of the Superset

Top-N	Superset		
	AVMiner	AVCLASS	AVCLASS2
1	0.800	0.867	0.161
2	0.910	0.867	0.774
3	0.935	0.867	0.802
4	0.966	0.867	0.860
5	0.979	0.867	0.884
6	0.981	0.867	0.890
7	0.989	0.867	0.890
8	0.989	0.867	0.891
9	0.990	0.867	0.891
10	0.991	0.867	0.902

According to the result, the accuracy of the previously mentioned sample "Spygames" increases a lot, up to 50%. The reason is that the token "Spy" appears in Drebin with richer context information, i.e., "Spybubble", "Spymob", "Spyphone", etc. Therefore, this has a positive effect on establishing the relation between "Spygames" and other tokens, which introduces the tokens "Spy" and "Games" into the *Best Cluster*.

D. Amount Sensitive

The previous results indicate that the performance of AVMiner may be sensitive to the number of malicious samples. With a larger amount, the diverse token context can provide much more contextual information, which will enhance the correlations between the related tokens. It will help AVMiner more easily to capture the token relations in the real world.

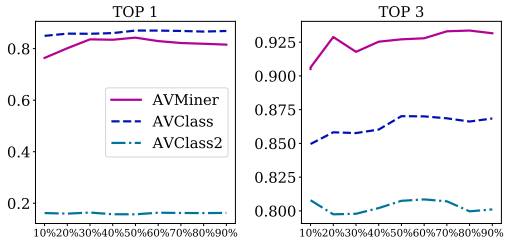

Fig. 6: Subset Evaluation Ranging from 10% to 90% of Superset

To validate our thought, we evaluate AVMiner on datasets of different sizes, ranging from 10% of the superset to 90% of the superset, comparing to the result by using the whole superset. Our tool aims to provide users with precise information with limited words. According to previous experiments, the accuracy becomes stable after TopN is larger than 3. So we choose Top1 and Top3 as representatives in the following experiments.

As shown in Figure 6, the subfigure on the left denotes the result of Top1, and the right side denotes the result of Top3. To ensure the fairness of sampling, we conduct these experiments 10 times each. We plot an area, and its upper bound and lower bound refer to the result range of the experiment of each subset. The line inside the area denotes the average result. We find that the size of the dataset indeed has some influence. The accuracy of AVMiner slightly drops when the dataset is relatively small. When the size of the dataset reaches 2,400 (about 30% of Superset), its accuracy doesn't differ much from the whole superset. Additionally, as the amount of data increases, the accuracy grows larger, and the fluctuation of accuracy also reduces. According to the result, the larger the data set, the richer the samples, the closer their distribution to the real world, and the more stable the results. Compared to AVCLASS and AVCLASS2, we regard the different data distributions as the root cause of the fluctuation.

To conclude, the experiment of negative sampling the Superset shows that the number of samples has some impact on the performance. Since the diversity of samples decreases due to the negative sampling, with the faded contextual information, some correlations between the tokens may become alienated, resulting in the result shown above.

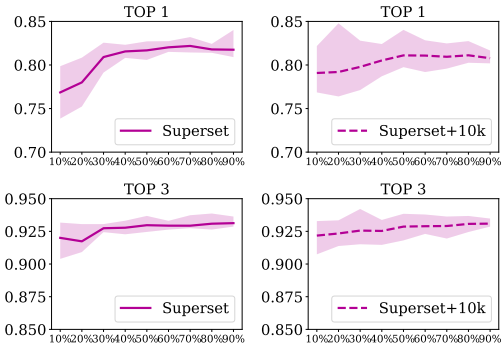

Fig. 7: Result comparison of sampling data and with external malicious samples on Top1 and Top3.

The subsets collected by sampling from the same dataset have a highly similar distribution to their superset. So the result is unpredictable when external data comes into the dataset and breaks the original distribution. To verify the robustness of AVMiner, we conduct an experiment on the subsets downsampling from the Superset, ranging from 10% to 90%, adding 10,000 malicious samples randomly collected from the real world (with no manually labeled ground truth). Due to the data source, we can only evaluate the result on the samples with ground truth, and we also repeat the experiments 10 times for sampling fairness.

As shown in the Figure 7, the left side of the figure denotes the Top1 and Top3 results of all subsets, and the right side of the figure denotes the Top1 and Top3 results of the subsets together with 10,000 randomly selected samples. When random external samples come in, it does lose some accuracy in Top1, but the overall trend is pretty close. According to Top3, the final average accuracy is almost identical. What's more, whether Top1 or Top3, when external data is introduced, the experimental results will be more volatile than the original data (have a wider range of colored areas). The experimental results confirm that different distributed data may lead to the loss of accuracy, but the overall stability is still strong.

Due to the space limitation, the sensitive study of *file format* and *time range* can be found in our arxiv version, the *large unlabeled data study* as well.

IV. Related Work

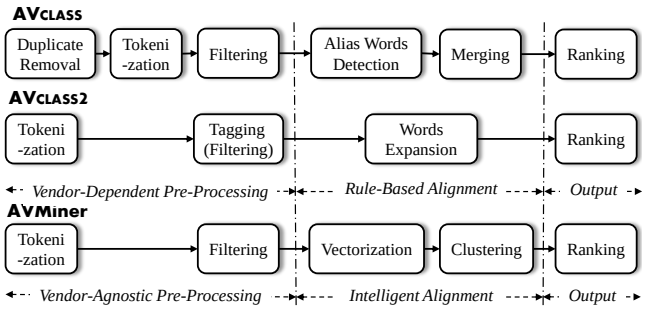

Fig. 8: Architecture comparison with previous works.

AV labels benefit the software security society for many applications, such as building malware dataset, malware detection and clustering[23], [24], [25], [26], [27]. Such as, Drebin[21] and Malheur[22] separately set up datasets on different platforms, and propose tools to detect them. However, AV labels are not always stable. Zhu et al.[5] survey on a large number of methods for malware labeling and analyze the problem of label inconsistency. They trace the AV labels of 14K samples for a year and conduct a thorough analysis indicating that there is an influence between vendors on producing AV labels. AV-Meter evaluates the performance of AV vendors on different datasets, and the presented results block researchers from using AV labels straightforwardly.

Malware Labeling. Facing the challenge above, AVMiner mainly focuses on mining the meaningful tokens from inconsistent AV labels produced by the third-party platform VirusTotal. We take it as an alternative to generate malware labels for the malicious samples, so do AVCLASS [6] and AVCLASS2 [8]. The procedures of the three methods are summarized and abstracted into three modules: pre-processing, alignment, and output. First, all of the methods process the AV labels into tokens minimizing the noise. Second, aligning the tokens in terms of format and semantics (merging the similar tokens, etc.) is ready for output. Finally, all the methods will output the result with a different ranking strategy for the users. The former works mainly leverage expert knowledge to set up the taxonomy, but AVMiner substitutes the procedures with intelligent ones. This design indeed causes more processing time, but it lights a way to label the malicious samples with less labor efficiency.

V. Conclusion

In this work, we propose AVMiner, an expansible AV label mining method. AVMiner takes as input a malicious sample with its corresponding AV labels, then pre-process, vectorize, group, and output the keywords. Without any expert knowledge, we are fully benefiting from the co-occurrence relations between tokens. According to the evaluation, AVMiner has better performance than previous works and shows robustness. And finally, we discuss the effectiveness of AVMiner on a large scale of malicious samples.

Acknowledgment

We sincerely thank the reviewers and anonymous shepherd for their valuable comments helping us to improve this work. This work was supported in part by grants from the Chinese National Natural Science Foundation (61272078, 62032010, 62172201), Science Foundation for Youths of Jiangsu Province (No. BK20220772), the program B for Outstanding PhD candidate of Nanjing University.

References

[1] "Av-test." https://www.av-test.org/en/, 2021.
[2] "Virustotal." https://www.virustotal.com/gui/, 2021.
[3] "Maec." https://maecproject.github.io/about-maec/, 2020.
[4] A. Mohaisen and O. Alrawi, "Av-meter: An evaluation of antivirus scans and labels," in *International conference on detection of intrusions and malware, and vulnerability assessment*, pp. 112–131, Springer, 2014.
[5] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online anti-malware engines," in *29th {USENIX} Security Symposium*, 2020.
[6] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 230–253, Springer, 2016.
[7] Y. Kurogome, "Avclass++: Yet another massive malware labeling tool," *Black Hat Europe*, 2019.
[8] S. Sebastián and J. Caballero, "Avclass2: Massive malware tag extraction from av labels," *proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2020.
[9] "Heartbleed." https://heartbleed.com/, 2020.
[10] "Eternalblue." https://www.avast.com/c-eternalblue, 2019.
[11] "Vjworm." https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/Worm.JS.VJWORM.AK/, 2021.
[12] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.
[13] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.
[14] L.-P. Jing, H.-K. Huang, and H.-B. Shi, "Improved feature selection approach tfidf in text mining," in *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2, pp. 944–946, IEEE, 2002.
[15] "Glove library." https://github.com/stanfordnlp/GloVe, 2021.
[16] "Gensim library." https://radimrehurek.com/gensim/, 2021.
[17] "scikit-learn library." https://scikit-learn.org/, 2021.
[18] "editdistance library." https://pypi.org/project/editdistance/0.3.1/, 2016.
[19] "tabulate library." https://pypi.org/project/tabulate/, 2021.
[20] "GNU Parallel." https://www.gnu.org/software/parallel/, 2010.
[21] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Ndss*, vol. 14, pp. 23–26, 2014.
[22] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.
[23] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering.," in *NDSS*, vol. 9, pp. 8–11, Citeseer, 2009.
[24] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 309–320, 2011.
[25] P. Kotzias, S. Matic, R. Rivera, and J. Caballero, "Certified pup: abuse in authenticode code signing," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 465–478, 2015.
[26] R. Perdisci, A. Lanzi, and W. Lee, "Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables," in *2008 Annual Computer Security Applications Conference (ACSAC)*, pp. 301–310, IEEE, 2008.
[27] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces.," in *NSDI*, vol. 10, p. 14, 2010.