AVCLASS: A Tool for Massive Malware Labeling

Marcos Sebastián[1], Richard Rivera[1 2], Platon Kotzias[1 2], and Juan Caballero[1]

[1] IMDEA Software Institute
[2] Universidad Politécnica de Madrid

Abstract. Labeling a malicious executable as a variant of a known family is important for security applications such as triage, lineage, and for building reference datasets in turn used for evaluating malware clustering and training malware classification approaches. Oftentimes, such labeling is based on labels output by antivirus engines. While AV labels are well-known to be inconsistent, there is often no other information available for labeling, thus security analysts keep relying on them. However, current approaches for extracting family information from AV labels are manual and inaccurate. In this work, we describe AVCLASS, an automatic labeling tool that given the AV labels for a, potentially massive, number of samples outputs the most likely family names for each sample. AVCLASS implements novel automatic techniques to address 3 key challenges: normalization, removal of generic tokens, and alias detection. We have evaluated AVCLASS on 10 datasets comprising 8.9 M samples, larger than any dataset used by malware clustering and classification works. AVCLASS leverages labels from any AV engine, e.g., all 99 AV engines seen in VirusTotal, the largest engine set in the literature. AVCLASS's clustering achieves F1 measures up to 93.9 on labeled datasets and clusters are labeled with fine-grained family names commonly used by the AV vendors. We release AVCLASS to the community.

Keywords: Malware Labeling, AV labels, Classification, Clustering

1 Introduction

Labeling a malicious executable as a variant of a known family is important for multiple security applications such as identifying new threats (by filtering known ones), selecting disinfection mechanisms, attribution, and malware lineage. Such labeling can be done manually by analysts, or automatically by malware classification approaches using supervised machine learning [8, 28, 29] (assuming the sample belongs to a family in the training set), and also through malware clustering approaches [2, 3, 24, 26] followed by a manual labeling process to assign a known family name to each cluster.

Labeling executables is also important for building reference datasets that are used by researchers for training those malware classification supervised approaches and for evaluating malware clustering results. This creates a bit of a chicken-and-egg problem, which prior work resolves by building reference datasets using AV labels [2, 3, 24, 28, 29]. However, AV labels are well-known to be inconsistent [2, 6, 18, 20]. In particular, AV engines often disagree on whether the same sample is malicious or not, and the family name that the label encodes may differ, among others, because of lack of a standard

naming convention (conventions such as CARO [7] and CME [4] are not widely used), lack of interest (the main goal of an AV is detection rather than classification [5, 10]), using heuristic or behavioral detections not specific to a family, and vendors assigning different names (i.e., *aliases*) to the same family.

Still, despite their known inconsistencies, AV labels are arguably the most common source for extracting malware labels. This likely happens because in many occasions no other ground truth is available, and because, despite its noisy nature, AV labels often contain the family name the analyst wants. Thus, extracting as accurate family information as possible from AV labels is an important problem.

Several limitations affect the process in which prior work builds family name ground truth from AV labels. First, some approaches use the full AV labels, which is inaccurate because the family name comprises only a fraction of the full label. For example, an AV engine may use different labels for samples in the same family, but still assign the same family name in those labels, e.g., when using two different detection rules for the family. Other works extract the family name in the labels through a manual process that is not detailed, does not handle aliases between family names, and does not scale to hundreds of thousands, or millions, of samples.

Second, it has been shown that no single AV engine detects all samples and that the number of AV engines needed to achieve high correctness in the family name is higher than for detection [20]. To address these issues, it is common to resort to majority voting among a fixed set of selected AV vendors. But, this requires selecting some AV vendors considered better at labeling, when prior work shows that some AV vendors may be good at labeling one family, but poor with others [20]. In addition, a majority cannot be reached in many cases, which means a family name cannot be chosen for those samples and they cannot be added into the evaluation or training data [25]. And, focusing on the samples where the majority agrees may bias results towards the easy cases [16]. Furthermore, prior work assumes the results of this process correspond to the ground truth, without quantitatively evaluating their quality.

In this work, we describe AVCLASS, an automatic labeling tool that given the AV labels for a, potentially massive, number of samples outputs the most likely family names for each sample, ranking each candidate family name by the number of AV engines assigning it to the sample. Selecting the top ranked family corresponds to a plurality vote, i.e., family with most votes wins. AVCLASS implements novel automatic techniques to address 3 main challenges: normalization, removal of generic tokens, and alias detection. Using those techniques AVCLASS automatically extracts as precise family information as possible from the input AV labels.

We envision AVCLASS being used in two main scenarios. In the first scenario, an analyst does not have access to a state-of-the-art malware clustering system (e.g., [2, 3, 24, 26]). When faced with labeling a large amount of samples, the analyst uses AVCLASS to efficiently obtain the most likely families for each sample. Here, AVCLASS acts as an efficient replacement for both clustering and labeling the resulting clusters.

In the second scenario, the analyst has access to an state-of-the-art malware clustering system and can use AVCLASS for 3 tasks. First, it can use AVCLASS to automatically label the output clusters with the most likely family name used by AV vendors.

Second, AVCLASS's output can be used to implement a feature based on AV labels (e.g., whether two samples have the same family name) that can be added to the existing clustering. Thus, rather than assuming that the AV labels constitute the ground truth, the analysts incorporates the AV labels knowledge into the clustering system. Third, AVCLASS's output can be used to build a reference clustering to evaluate the clustering results. Since AVCLASS tags each candidate family name with a confidence factor based on the number of AV engines using the name, the analyst can select a threshold on the confidence factor for building the reference dataset, e.g., replacing the default plurality vote with a more conservative (majority or else) vote.

The salient characteristics of AVCLASS are:

- **Automatic.** AVCLASS removes manual analysis limitations on the size of the input dataset. We have evaluated AVCLASS on 8.9 M malicious samples, larger than any dataset previously used by malware clustering and classification approaches.
- **Vendor-agnostic.** Prior work operates on the labels of a fixed subset of 1–48 AV engines. In contrast, AVCLASS operates on the labels of any available set of AV engines, which can vary from sample to sample. All labels are used towards the output family name. AVCLASS has been tested on all 99 AV engines we observe in VirusTotal [31], the largest AV engine set considered so far.
- **Plurality vote.** AVCLASS performs a plurality vote on the normalized family names used by all input engines. A plurality vote outputs a family name more often than a majority vote, since it is rare for more than half of the AV engines to agree.
- **Cross-platform.** AVCLASS can cluster samples for any platforms supported by the AV engines. We evaluate AVCLASS on Windows and Android samples.
- **Does not require executables.** AV labels can be obtained from online services like VirusTotal using a sample's hash, even when the executable is not available.
- **Quantified accuracy.** The accuracy of AVCLASS has been evaluated on 5 publicly available malware datasets with ground truth, showing that it can achieve an F1 score of up to 93.9.
- **Reproducible.** We describe AVCLASS in detail and release its source code.[3]

2 Related Work

Table 1 summarizes relevant works that have used AV labels. For each work, it first presents the year of publication and the goal of the work, which can be malware detection, clustering, classification, combinations of those, cluster validity evaluation, as well as the development of metrics to evaluate AV labels. Then, it describes the granularity of the information extracted from the AV labels, which can be Boolean detection (i.e., existence or absence of the label), coarse-grained classification in particular if a sample is a potentially unwanted program (PUP) or malware, and extracting the family name. Next, it shows the number of labeled samples used in the evaluation and the number

[3] https://github.com/malicialab/avclass

Table 1: Related work that uses AV labels. The number of samples includes only those labeled using AV results and for classification approaches only malicious samples.

			Granularity							
Work	**Year**	**Goal**	**Det.**	**PUP**	**Fam.**	**Samples**	**AV**	**Eval**	**Train**	**Norm**
Bailey et al. [2]	2007	Cluster	✓	✗	✓	8.2K	5	✓	✗	✗
Rieck et al. [28]	2008	Classify	✓	✗	✓	10K	1	✓	✓	✗
McBoost [23]	2008	Detect	✓	✗	✗	5.5K	3	✓	✓	✗
Bayer et al. [3]	2009	Cluster	✓	✗	✓	75.7K	6	✓	✗	✗
Perdisci et al. [24]	2010	Cluster+Detection	✓	✗	✓	25.7K	3	✓	✗	✓
Malheur [29]	2011	Cluster+Classify	✓	✗	✓	3.1K	6	✓	✓	✗
BitShred [13]	2011	Cluster	✓	✗	✓	3.9K	40	✓	✗	✓
Maggi et al. [18]	2011	Metrics	✓	✗	✓	98.8K	4	✗	✗	✓
VAMO [25]	2012	Cluster Validity	✓	✗	✓	1.1M	4	✓	✗	✓
Rajab et al. [27]	2013	Detect	✓	✗	✗	2.2K	45	✓	✗	✗
Dahl et al. [8]	2013	Detect+Classify	✓	✗	✓	1.8M	1	✓	✓	✗
Drebin [1]	2014	Detect	✓	✓	✓	5.5K	10	✓	✓	✓
AV-Meter [20]	2014	Metrics	✓	✗	✓	12 K	48	✗	✗	✗
Malsign [15]	2015	Cluster	✓	✓	✗	142K	11	✓	✗	✗
Kantchelian et al. [14]	2015	Detect	✓	✗	✗	279K	34	✓	✓	✗
Miller et al. [19]	2016	Detect	✓	✗	✗	1.1M	32	✓	✓	✗
MtNet [11]	2016	Detect+Classify	✓	✗	✓	2.8M	1	✓	✓	✗
Hurier et al. [12]	2016	Metrics	✓	✓	✓	2.1M	66	✗	✗	✗
AVCLASS	2016	Cluster+Label	✓	✓	✓	8.9M	99	✓	✗	✓

of vendors those labels come from. For supervised approaches, the samples column includes only malicious samples in the training set. As far as we know the largest dataset used previously for malware clustering or classification is by Huang and Stokes [11], which comprises 6.5 M samples: 2.8 M malicious and 3.7 M benign. In contrast, we evaluate AVCLASS on 8.9 M malicious samples, making it the largest so far. The next two columns capture whether the AV labels are used to evaluate the results and for training machine learning supervised approaches. Finally, the last column captures if normalization is applied to the AV labels (✓), or alternatively the full label is used (✗).

Most works consider a sample malicious if at least a threshold of AV engines detects it (i.e., returns a label) and weigh each AV engine equally in the decision. There is no agreement in the threshold value, which can be a single AV engine [25, 32], two [1], four [15, 19], or twenty [13]. Some works evaluate different thresholds [20, 22] showing that small threshold increases, quickly reduce the number of samples considered malicious. Recently, Kantchelian et al. [14] propose techniques for weighing AV engines differently. However, they assume AV labels are independent of each other, despite prior work having found clusters of AV engines that copy the label of a leader [20], which we also observe. In addition, an AV engine with poor overall labels may have highly accurate signatures for some malware families. In our work we adjust the influence of AV engines that copy labels and then weigh remaining labels equally.

Other works show that AVs may change their signatures over time, refining their labels [9, 14]. Recently, Miller et al. [19] argue that detection systems should be trained not with the labels available at evaluation time (e.g., latest VT reports), but with the

labels available at training time. Otherwise, detection rate can be inflated by almost 20 percentage points. However, for evaluating clustering results, it makes sense to use the most recent (and refined) labels.

AV label inconsistencies. Prior work has identified the problem of different AV engines disagreeing on labels for the same sample [2, 6, 18, 20]. While such discrepancies are problematic, security analysts keep coming back to AV labels for ground truth. Thus, we believe the key question is how to automatically extract as much family information as possible from those labels and to quantitatively evaluate the resulting reference dataset. We propose an automatic labeling approach that addresses the most important causes for discrepancies, namely different naming schemes, generic tokens, and aliases.

Li et al. [16] analyze the use of a reference clustering extracted from AV labels to evaluate clustering results. They argue that using only a subset of samples, for which the majority of AV engines agrees, biases the evaluation towards easy-to-cluster samples. AVCLASS automatically extracts the most likely family names for a sample (even if no majority agrees on it), helping to address this concern by enlarging the reference dataset. Mohaisen and Alrawi [20] propose metrics for evaluating AV detections and their labels. They show how multiple AVs are complementary in their detection and also that engines with better detection rate do not necessarily have higher correctness in their family names. Recently, Hurier et al. [12] propose further metrics to evaluate ground truth datasets built using AV labels. One limitation of proposed metrics is that they operate on the full AV labels without normalization.

Most related to our work is VAMO [25], which proposes an automated approach for evaluating clustering results. VAMO normalizes the labels of 4 AV vendors to build an AV graph (introduced in [24]) that captures the fraction of samples where labels, possibly from different engines, appear together. Our alias detection approach is related, although VAMO does not output aliases as AVCLASS does. Furthermore,VAMO finds the set of reference clusters from the AV labels that best agrees with a third-party clustering, while AVCLASS labels samples without requiring third-party clustering results.

Naming conventions. There have been attempts at reducing confusion in malware labels through naming conventions, but they have not gained much traction. A pioneering effort was the 1991 CARO Virus Naming Convention [7]. More recently, the Common Malware Enumeration (CME) Initiative [4] provides unique identifiers for referencing the same malware across different names.

3 Approach

Figure 1 shows the architecture of AVCLASS. It comprises two phases: *preparation* and *labeling*. During the preparation phase, an analyst runs the *generic token detection* and *alias detection* modules on the AV labels of a large number of samples to produce lists of generic tokens and aliases, which become inputs to the labeling phase. In particular, the generic token detection takes as input the AV labels of samples for which their family is known (e.g., from publicly available labeled datasets [1, 21, 29, 33]) and outputs a list of generic tokens, i.e., tokens that appear in labels of samples from different fami-

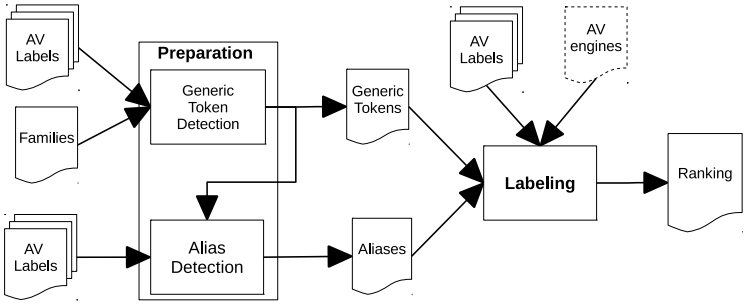

Fig. 1: AVCLASS architecture.

lies. The alias detection module takes as input AV labels of a large number of unlabeled samples and outputs pairs of family names that alias to each other.

The labeling phase is the core of AVCLASS and implements the label normalization process. It takes as input the AV labels of a large number of samples to be labeled, a list of generic tokens, a list of aliases, and optionally a list of AV engines to use. For each sample to be labeled, it outputs a ranking of its most likely family names. The list of generic tokens and aliases are the outputs of the preparation phase. By default, AVCLASS uses all AV engines in the set of AV labels for a sample. However, by providing an AV engine list, the analyst can restrict the processing to labels from those engines.

AVCLASS implements both the preparation and labeling phases. But, we expect that many analysts may not have large numbers of samples used for preparation. Thus, AVCLASS also includes default lists of generic tokens and aliases obtained in this work, so that an analyst can skip the preparation phase.

The remainder of this section first details the labeling phase (Section 3.1), and then the generic token detection (Section 3.2) and alias detection (Section 3.3) preparation modules. To illustrate the approach, we use the running example in Figure 2.

3.1 Labeling

The labeling phase takes as input the AV labels of a, potentially massive, number of samples. For each sample, it returns a ranking of its most likely family names. This Section describes the 8 steps used in labeling each sample.

AV selection (optional). By default, AVCLASS processes the labels of all AV engines in the input set of a sample. The labels of each sample may come from a different set of AV engines. This design decision was taken because selecting a fixed set of AV engines (as done in prior work) is difficult since there is no real information about which engines are better, and some engines may be good in a family but poor with others. Furthermore, a fixed set of AV engines throws away information as certain AV vendors may have been analyzed only by certain AV vendors. In particular, it is common to obtain AV labels from VT, which has used different AV engines to scan uploaded files over time. Overall, we observe 99 different AV engines in our VT reports, which are detailed in Table 11

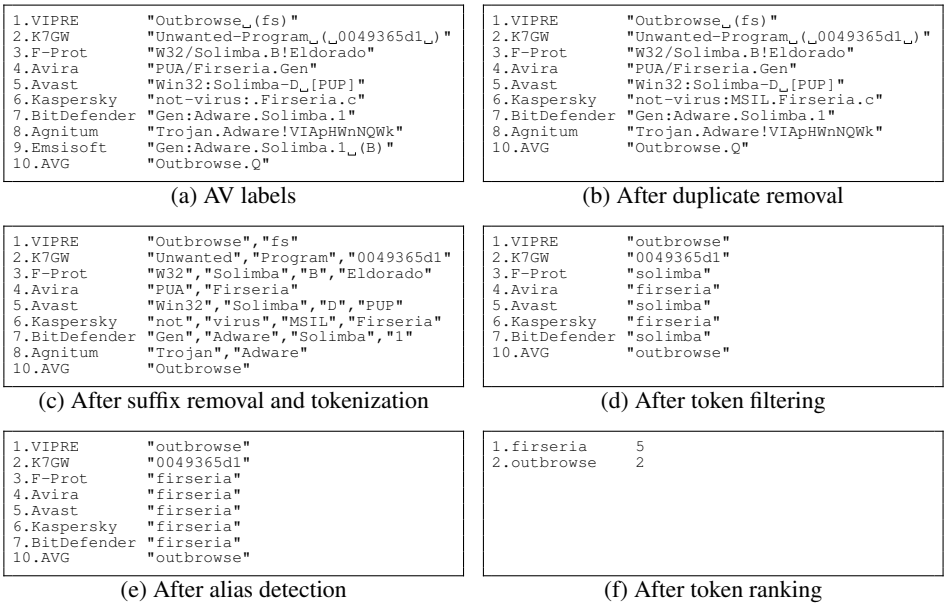

Fig. 2: Running example.

in the Appendix. Some engines are only seen for a few days, while others have been continually used by VT for nearly a decade.

Still, an analyst can optionally provide an input list of engines to AVCLASS. If provided, labels from engines not in the input list are removed at this step and only labels from the input set of AV engines are used for every sample. For brevity, our running example in Figure 2 assumes the analyst provided an input list with 10 engines. Figure 2a shows the input AV labels from the selected 10 engines for the same sample.

Duplicate removal. The same AV vendor may have multiple engines such as McAffee and McAffee-GW-Edition, or TrendMicro and TrendMicro-HouseCall. Those engines often copy labels from each other. While we could include only one engine per vendor, the reality is that their detections often differ. In addition, we observe groups of AV vendors that copy labels from each other, something also observed in prior work [20]. In both situations, the detection from these groups of engines are not independent (an assumption of some prior works [14]).

To avoid giving too much weight on the selected family name to vendors with multiple engines, or whose labels are often copied, we leverage the observation that when two AV engines output exactly the same label for a sample this very likely corresponds to one of those two situations. This happens because each vendor structures its labels differently and also uses slightly different keywords in their labels, so that two engines producing exactly the same label is rare unless they are copying each other. Thus, at this step, AVCLASS remove all duplicate labels. A special case is a vendor (*Emsisoft*) that when copying labels adds to them the suffix " (B)". For this vendor, we first remove this suffix from its labels, and then check for duplicates. We have not observed

any other such cases. Figure 2b shows how the *Emsisoft* label is removed at this step as a duplicate of the *BitDefender* label.

Suffix removal. We have empirically observed that most noise in AV labels is introduced in the suffix, i.e., the part of the AV label after the family name, where AV vendors may encode information such as rule numbers and hexadecimal strings that may be hashes. In general, it is difficult to remove those suffixes for all engines as vendors use different label structures, which may even change over time. Still, we have found 3 simple rules to truncate useless suffixes: (1) for 17 AV engines, truncate label after last dot; (2) for *AVG*, truncate after last dot if the suffix only contains digits or uppercase chars; and (3) for *Agnitum*, truncate after the last '!' character. Suffix removal is the only engine-specific step in AVCLASS.

Tokenization. The next step is to split each label into tokens. We use a simple tokenization rule that splits the label on any sequence of consecutive non-alphanumeric characters. Figure 2c shows the results of the suffix removal and tokenization steps. Labels 4, 6, 8, and 10 have been truncated by the suffix removal rules, and all labels have been tokenized.

Token filtering. The goal of this step is to remove tokens that are not family names. Each token goes through five substeps: (1) convert to lowercase; (2) remove digits at the end of the token; (3) remove token if short, i.e., less than 4 characters; (4) remove token if present in the input list of generic tokens; and (5) remove token if it is a prefix of the sample's hash[4]. Figure 2d shows the results of token filtering where label 8 was removed as a result of not having any tokens left.

Alias replacement. Different vendors may use different names for the same family, i.e., aliases. If a token shows in the input list of aliases as being an alias for another family name, the token is replaced by the family name it aliases. The alias detection process is detailed in Section 3.3. Figure 2d shows the results after alias replacement, where token *solimba* has been identified as an alias for the *firseria* family.

Token ranking. Next, tokens are ranked by decreasing number of engines that include the token in their label. Tokens that appear in at most one AV engine are removed. This allows removing random tokens that earlier steps may have missed, as the likelihood is low that a random token appears in labels from multiple AV engines that did not copy their labels. At this point, the ranking captures the candidate family names for the sample and the number of AV engines that use each token can be seen as a confidence score. Figure 2f shows the final token ranking for our running example where token *0049365d1* have been removed because it appears in only one label.

Family selection. AVCLASS chooses the most common token (top of the ranking) as the family name for the sample. This corresponds to a plurality vote on the candidate family names. AVCLASS also has a verbose option to output the complete ranking, which is useful to identify samples with multiple candidate family names with close scores, which may deserve detailed attention by the analyst. In our running example, the selected family is *firseria*, which outscores 5 to 2 the other possible family name.

[4] We check the sample's MD5, SHA1, and SHA256 hashes.

Table 2: Categories in the manual generic token list.

Category	Tokens	Example Tokens
Architecture	14	android, linux, unix
Behavior: download	29	download, downware, dropped
Behavior: homepage modification	2	homepage, startpage
Behavior: injection	5	inject, injected, injector
Behavior: kill	5	antifw, avkill, blocker
Behavior: signed	2	fakems, signed
Behavior: other	3	autorun, proxy, servstart
Corrupted	2	corrupt, damaged
Exploit	2	expl, exploit
File types	15	html, text, script
Generic families	13	agent, artemis, eldorado
Heuristic detection	12	generic, genmalicius, heuristic
Macro	11	badmacro, macro, x2km
Malicious software	5	malagent, malicious, malware
Malware classes	53	spyware, trojan, virus
Misc	9	access, hack, password
Packed	17	malpack, obfuscated, packed
Packer	6	cryptor, encoder, obfuscator
Patch	3	patched, patchfile, pepatch
Program	5	application, program, software
PUP	29	adware, pup, unwanted
Suspicious	13	suspected, suspicious, variant
Test	2	test, testvirus
Tools	8	fraudtool, tool, virtool
Unclassified	3	unclassifiedmalware, undef, unknown

3.2 Generic Token Detection

AV labels typically contain multiple generic tokens not specific to a family. For example, the labels in Figure 2 include generic tokens indicating, among others, the sample's architecture (e.g., *Win32*, *Android*), that a sample is unwanted (e.g., *Unwanted*, *Adware*, *PUP*), generic malware classes (e.g., *Trojan*), and generic families used with heuristic rules (e.g., *Eldorado*, *Artemis*). The generic token detection module is used during the preparation phase to automatically build a list of generic tokens used as input to the labeling phase in Section 3.1.

The intuition behind our technique for identifying generic tokens is that tokens appearing in the labels of samples known to be of different families cannot be specific to a family, and thus are generic. For example, an AV engine may output the label *Gen:Adware.Firseria.1* for a sample known to be of the Firseria adware family and the label *Gen:Adware.Outbrowse.2* for a sample known to be of the Outbrowse adware family. Here, tokens *Gen* and *Adware* are likely generic because they are used with samples of different families, and thus are not specific to the Firseria or Outbrowse families.

The generic token detection module takes as input samples for which their family name is known. It iterates on the list of input samples. For each sample, it builds a *sample token list*, by iterating on the set of AV labels for the sample. For each label, it

tokenizes the label on non-alphanumeric characters, converts tokens to lowercase, removes digits at the end of the token, removes tokens less than 4 characters, and appends the remaining tokens to the sample token list. Once all labels are processed, it removes duplicate tokens from the sample token list. The sample token list for the sample in Figure 2 would be: *outbrowse*, *unwanted*, *program*, *0049365d1*, *solimba*, *eldorado*, *firseria*, *virus*, *msil*, *adware*, and *trojan*. Then, it iterates on the tokens in the sample token list updating a *token family map*, which maps each unique token to the list of families of the samples where the token appears in their labels.

After all samples have been processed, it iterates on the token family map. Each token that does not match a family name and has a count larger than T_{gen} is considered generic. The default $T_{gen} > 8$ threshold is chosen empirically in Section 4.3. For example, tokens *firseria* and *solimba* may have appeared only in labels of samples from the *Firseria* family and thus are not generic, but token *eldorado* may have appeared in labels from samples of 9 different families and is identified as generic.

We have applied this approach to automatically generate a list of generic tokens. One author has also manually generated a list of generic tokens. Our experiments in Section 4.3 show that the automatically generated generic token list performs similarly in most cases, and even outperforms the manually generated lists in some cases, while scaling and being independent of an analyst's expertise.

Table 2 shows the 15 categories of generic tokens in the manually built generic token list. For each category, it shows the number of tokens in the category and some example tokens. The categories show the wealth of information that AV vendors encode in their labels. They include, among others, architectures; behaviors like homepage modification, code injection, downloading, and disabling security measures; file types; heuristic detections; macro types; malware classes; encrypted code; and keywords for potentially unwanted programs. The categories with more generic tokens are malware classes with 53 tokens (e.g., *trojan*, *virus*, *worm*, *spyware*), download behavior with 29 (e.g., *download*, *dload*, *downl*, *downware*), and potentially unwanted programs with 29 (e.g., *pup*, *adware*, *unwanted*).

3.3 Alias Detection

Different vendors may assign different names (i.e., aliases) for the same family. For example, some vendors may use *zeus* and others *zbot* as aliases for the same malware family. The alias detection module is used during the preparation phase to automatically build a list of aliases used as input to the labeling phase in Section 3.1.

The intuition behind our technique for automatically detecting aliases is that if two family names are aliases, they will consistently appear in the labels of the same samples. Alias detection takes as input the AV labels of a large set of samples, for which their family does not need to be known, and a generic token list. Thus, alias detection runs after the generic token detection, which prevents generic tokens to be detected as aliases. Alias detection outputs a list of (t_i, t_j) token pairs where t_i is an alias for t_j. This indicates that t_i can be replaced with t_j in the alias detection step in Section 3.1.

Table 3: Top 10 families by number of aliases.

Family	Aliases	Example Aliases
wapomi	12	pikor, otwycal, protil
firseria	10	firser, popeler, solimba
vobfus	9	changeup, meredrop, vbobfus
virut	8	angryangel, madangel, virtob
gamarue	7	debris, lilu, wauchos
hotbar	7	clickpotato, rugo, zango
bandoo	6	ilivid, seasuite, searchsuite,
gamevance	6	arcadeweb, gvance, rivalgame
loadmoney	6	ldmon, odyssey, plocust
zeroaccess	6	maxplus, sirefef, zaccess

Alias detection iterates on the list of input samples. For each sample, it builds the sample token list in the same manner as described in the generic token detection in Section 3.2, except that tokens in the generic token list are also removed. Then, it iterates on the tokens in the sample token list updating two maps. It first increases the *token count map*, which stores for each unique token the number of samples where the token has been observed in at least one label. Then, for each pair of tokens in the sample token list it increases the *pair count map* that stores for each token pair the number of samples in which those two tokens have been observed in their labels.

We define the function $alias(t_i, t_j) = \frac{|(t_i,t_j)|}{|t_i|}$, which captures the fraction of times that the pair of tokens (t_i, t_j) appears in the same samples. The numerator can be obtained from the pair count map and the denominator from the token count map. Note that $alias(t_i, t_j) \neq alias(t_j, t_i)$.

Once all samples have been processed, the alias detection iterates on the pair count map. For each pair that has a count larger than n_{alias}, it computes both $alias(t_i, t_j)$ and $alias(t_j, t_i)$. If $alias(t_i, t_j) > T_{alias}$ then t_i is an alias for t_j. If $alias(t_j, t_i) > T_{alias}$ then t_j is an alias for t_i. If both $alias(t_i, t_j) > T_{alias}$ and $alias(t_j, t_i) > T_{alias}$ then the less common token is an alias for the most common one.

The two parameters are empirically selected in Section 4.4 to have default values $n_{alias} = 20$ and $T_{alias} = 0.94$. n_{alias} is used to remove pairs of tokens that have not been seen enough times, so that a decision on whether they are aliases would have low confidence. T_{alias} controls the percentage of times the two tokens appear together. For t_j to be an alias for t_i, t_j should appear in almost the same samples where t_i appears, but T_{alias} is less than one to account for naming errors.

Table 3 shows the Top 10 families by number of aliases. For each alias, it shows the chosen family name, the total number of aliases for that family, and some example aliases that appear both in the automatically and manually generated alias lists.

Table 4: Datasets used in evaluation.

Dataset	Platform	Lab.	Samples	EXE	Collection Period
University	Windows	✗	7,252,810	✗	01/2011 - 08/2015
Miller et al. [19]	Windows	✗	1,079,783	✗	01/2012 - 06/2014
Andrubis [17]	Android	✗	422,826	✗	06/2012 - 06/2014
Malsign [15]	Windows	✓	142,513	✗	06/2012 - 02/2015
VirusShare_20140324 [30]	Android	✗	24,317	✓	05/2013 - 05/2014
VirusShare_20130506 [30]	Android	✗	11,080	✓	06/2012 - 05/2013
Malicia [21]	Windows	✓	9,908	✓	03/2012 - 02/2013
Drebin [1]	Android	✓	5,560	✓	08/2010 - 10/2012
Malheur Reference [29]	Windows	✓	3,131	✗	08/2009 - 08/2009
MalGenome [33]	Android	✓	1,260	✓	08/2008 - 10/2010

4 Evaluation

4.1 Datasets

We evaluate AVCLASS using 10 datasets summarized in Table 4. The table shows the architecture (5 Windows, 5 Android), whether the samples are labeled with their known family name, the number of samples in the dataset, whether the binaries are publicly available (otherwise we only have their hashes), and the collection period. In total, the datasets contain 8.9 M distinct samples collected during 7 years (08/2008 - 08/2015). Some of the datasets overlap, most notably the Drebin [1] dataset is a superset of MalGenome [33]. We do not remove duplicate samples because this way it is easier for readers to map our results to publicly available datasets.

Labeled datasets. All 5 labeled datasets come from prior works [1, 15, 21, 29, 33]. Among the 3 labeled Windows datasets, Malheur and Malicia contain only malware samples. In contrast, the Malsign dataset [15] contains majoritarily PUP. Each of the labeled datasets went through 2 processes: clustering and labeling. Samples may have been clustered manually (MalGenome), using AV labels (Drebin), or with automatic approaches (Malheur, Malicia, Malsign). For labeling the output clusters, the authors may have used AV labels (Drebin), manual work (MalGenome), the most common feature value in the cluster (Malsign), or a combination of popular features values and information from public sources (Malicia). Drebin [1] is a detection approach and the family classification was done separately using AV labels. Because of this we later observe best results of AVCLASS on this dataset.

Drebin, MalGenome, Malheur, and Malicia datasets are publicly available. Thus, AV vendors could have refined their detection labels using the dataset clustering results after they became public. In contrast, the Malsign dataset and thus its clustering results (i.e., labels) are not publicly available.

Unlabeled datasets. For the unlabeled datasets, we do not know the family of the samples and in some cases we only have access to the hashes of the samples, but not their binaries. The University dataset contains malware hashes collected from different sources including a commercial feed. It is our largest dataset with 7.2 M samples. The Andru-

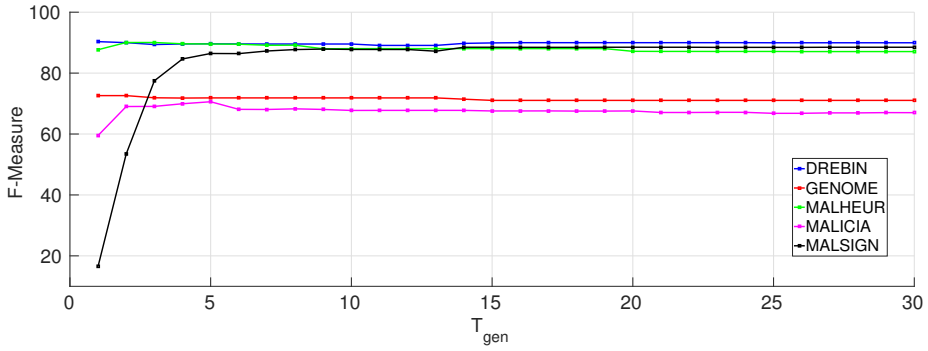

Fig. 3: Parameter selection for generic token detection.

bis dataset [17] contains hashes of samples submitted by users to be analyzed by the Andrubis online service. The two VirusShare [30] and the Miller et al. [19] datasets are publicly available.

For all samples in the 10 datasets we were able to collect a VT report. The VT report collection started on September 2015 and took several months. Overall, we observe 99 AV engines in the VT reports.

4.2 Metrics

To evaluate the accuracy of AVCLASS, we use an external clustering validation approach that compares AVCLASS's clustering results with a reference clustering from one of the datasets in Table 4 for which we have ground truth. Note that the external validation evaluates if both clusterings group the samples similarly. It does not matter if the family names assigned to the equivalent cluster in both clustering differ. If AVCLASS is not able to find a family name for a sample (e.g., because all its labels are generic), the sample is placed in a singleton cluster. Similar to prior work [3, 15, 21, 25, 29] we use the precision, recall, and F1 measure metrics, which we define next.

Let M be a malware dataset, $R = \{R_1, ..., R_s\}$ be the set of s *reference clusters* from the dataset's ground truth, and $C = \{C_1, ..., C_n\}$ be the set of n clusters output by AVCLASS over M. In this setting, precision, recall, and F1 measure are defined as

- **Precision.** $Prec = 1/n \cdot \sum_{j=1}^{n} max_{k=1,...,s}(|C_j \bigcap R_k|)$
- **Recall.** $Rec = 1/s \sum_{k=1}^{s} max_{j=1,...,n}(|C_j \bigcap R_k|)$
- **F-measure Index.** $F1 = 2\frac{Prec \cdot Rec}{Prec + Rec}$

4.3 Generic Token Detection

The generic token detection, detailed in Section 3.2, takes as input the AV labels for samples with family name and counts the number of families associated to each remaining token after normalization. Tokens that appear in more than T_{gen} families are

considered generic. To select the default threshold, we produce generic token lists for different T_{gen} values and evaluate the accuracy of the labeling phase using those generic token lists. Figure 3 shows the F1 measure as T_{gen} increases for datasets with ground truth. Based on Figure 3 results, we select $T_{gen} > 8$ as the default threshold. The rest of experiments use, unless otherwise noted, the automatically generated generic token list with this default threshold, which contains 288 generic tokens. In comparison the generic token list manually generated by one author contains 240 generic tokens.

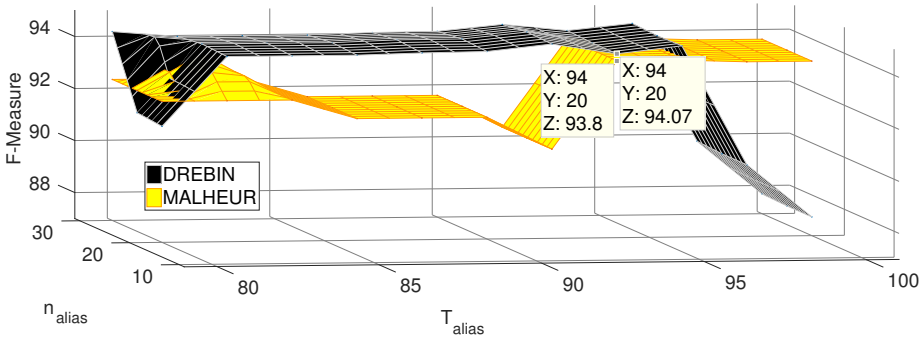

Fig. 4: Parameter selection for alias detection.

4.4 Alias Detection

The alias detection module, detailed in Section 3.3, requires two parameters: n_{alias} and T_{alias}. To select their default values, we first produce alias lists for different combinations of those parameters using as input the 5 datasets with unlabeled samples. Then, we evaluate the accuracy of the labeling phase using those alias lists. Figure 4 shows the F_1 measure for different combinations of parameter values on the Drebin and Malheur datasets. The parameter values that maximize the mean value in both surfaces are $n_{alias} = 20$ and $T_{alias} = 0.94$. The rest of experiments use, unless otherwise noted, the automatically generated alias list with these default values, which contains 4,332 alias pairs. In comparison, the alias list manually generated by one author contains 133 alias pairs.

4.5 Evaluation on Labeled Datasets

In this section we evaluate the accuracy of AVCLASS on the labeled datasets. We first compare the reference clustering provided by the dataset labels with the clustering output by AVCLASS (i.e., samples assigned the same label by AVCLASS are in the same cluster) using the precision, recall, and F1 measure metrics introduced in Section 4.2. Then, we examine the quality of the output labels.

Clustering accuracy. Table 5 summarizes the clustering accuracy results for 3 scenarios. *Full Label* corresponds to not using AVCLASS but simply doing a plurality vote

Table 5: Accuracy evaluation. *Full Label* corresponds to using a plurality vote on all labels without normalization. *Manual* corresponds to running AVCLASS with manually generated generic token and alias lists. AVCLASS corresponds to running AVCLASS with automatically generated generic token and alias lists. The MalGenome* row corresponds to grouping the 6 DroidKungFu variants in MalGenome into a single family.

	AVCLASS			**Manual**			**Full Label**		
Dataset	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
Drebin	95.2	92.5	93.9	95.4	88.4	91.8	92.9	40.7	56.6
Malicia	95.3	46.3	62.3	94.9	68.0	79.2	98.6	2.4	4.6
Malsign	96.3	67.0	79.0	90.4	90.7	90.5	88.7	15.9	26.9
MalGenome	67.5	98.8	80.2	68.3	93.3	78.8	99.5	79.4	88.3
MalGenome*	87.2	98.8	92.6	87.9	93.3	90.5	99.7	63.3	77.5
Malheur	89.3	93.8	91.5	90.4	98.3	94.2	96.3	74.8	84.2

on the full AV labels without normalization. *Manual* corresponds to running AVCLASS with manually generated generic token and alias lists. AVCLASS corresponds to running AVCLASS with automatically generated generic token and alias lists.

The results show that using AVCLASS increases the F1 measure compared to using the full label in 4 datasets (Drebin, Malicia, Malsign, and Malheur). The median F1 measure improvement is 37 F1 measure percentual points and can reach 13 times higher (Malicia). The exception is the MalGenome dataset, whose F1 measure decreases. Manual examination shows that the main problem is that the MalGenome dataset differentiates 6 variants of the DroidKungFu family (DroidKungFu1, DroidKungFu2, DroidKungFu3, DroidKungFu4, DroidKungFuSapp, DroidKungFuUpdate). However, AV labels do not capture version granularity and label all versions as the same family. If we group all 6 DroidKungFu variants into a single family (MalGenome* row in Table 5), the F1 measure using AVCLASS increases 12 points (from 80.2 to 92.6) and the full label results decreases 11 points (from 88.3 to 77.5). This shows that AV labels are not granular enough to identify specific family versions.

Comparing the *Manual* section of Table 5 with the AVCLASS section shows that the automatically generated lists of generic tokens and aliases work better in 2 datasets (MalGenome and Drebin) and worse in 3 (Malicia, Malsign, Malheur). For Malheur the difference is small (2.7 F1 points), but for Malicia and Malsign it reaches 11-17 F1 points. Overall, the automatically generated lists have comparable accuracy to the manual ones, although an analyst can improve results in some datasets. While the manual list raises the worst F1 measure from 62.3 to 79.2, the automatic generation is faster, more convenient, and does not depend on the analyst's skill. To combine scalability with accuracy, an analyst could first produce automatically the lists and then refine them manually based on his expertise.

The final F1 measure for AVCLASS with automatically generated lists of generic tokens and aliases ranges from 93.9 for Drebin down to 62.3 for Malicia. The higher accuracy for Drebin is due to that dataset having been manually clustered using AV labels. The lower accuracy for Malicia is largely due to *smartfortress* likely being an

Table 6: Labels for the top 5 clusters identified by AVCLASS in the Miller et al. dataset and the most common full labels on the same dataset.

(a) AVCLASS.

#	Label	Samples
1	vobfus	58,385
2	domaiq	38,648
3	installrex	37,698
4	firseria	28,898
5	multiplug	26,927

(b) Full labels.

#	Label	Samples
1	Trojan.Win32.Generic!BT	42,944
2	Win32.Worm.Allaple.Gen	12,090
3	Gen:Variant.Adware.Graftor.30458	10,844
4	Gen:Adware.MPlug.1	10,332
5	Trojan.Generic.6761191	8,986

Table 7: Clustering results on unlabeled datasets.

Dataset	Samples	Clusters	Singletons	Unlab.	Largest	Runtime
University	7,252,810	1,465,901	1,456,375	19.2%	701,775	235 min. 33s
Miller et al.	1,079,783	187,733	183,804	16.6%	56,044	35 min. 42s
Andrubis	422,826	7,015	6,294	1.3%	102,176	12 min. 47s
VirusShare_20140324	24,317	2,068	1,844	6.9%	7,350	48s
VirusShare_20130506	11,080	568	446	3.3%	3,203	17s

(undetected) alias for the *winwebsec* family. Manually adding this alias improves the F1 measure by 18 points. The reason for the large impact of this alias is that the Malicia dataset is strongly biased towards this family (59% of samples are in family *winwebsec*).

Label quality. The clustering evaluation above focuses on whether samples are grouped by AVCLASS similarly to the ground truth, but it does not evaluate the quality of the family names AVCLASS outputs. Quantifying the quality of the family names output by AVCLASS is challenging because the ground truth may contain manually selected labels that do not exactly match the AV family names. Table 6 shows on the left the labels assigned to the top 5 clusters in the Miller dataset by AVCLASS and on the right, the labels for the top 5 clusters when the full AV labels are used. The table shows that the cluster labels automatically produced by AVCLASS are more fine-grained thanks to the generic token detection, and also assigned to a larger number of samples thanks to the normalization and alias detection techniques. More examples of the final labels output by AVCLASS are shown in Table 8, which is discussed in the next section.

4.6 Evaluation on Unlabeled Datasets

In this section we apply AVCLASS to label samples in datasets without ground truth. Table 7 summarizes the clustering results of using AVCLASS with automatically generated lists on the 5 unlabeled datasets. For each dataset it shows: the number of samples being clustered, the number of clusters created, the number of singleton clusters with only one sample, the percentage of all samples that did not get any label, the size of the largest cluster, and the labeling runtime. The results show that 78%–99% of the clusters

Table 8: Top 10 clusters on unlabeled datasets.

(a) University.

#	Label	Samples
1	vobfus	701,775
2	multiplug	669,596
3	softpulse	473,872
4	loadmoney	211,056
5	virut	206,526
6	toggle	108,356
7	flooder	96,571
8	zango	89,929
9	upatre	82,413
10	ibryte	80,923

(b) Miller.

#	Label	Samples
1	vobfus	58,385
2	domaiq	38,648
3	installrex	37,698
4	firseria	28,898
5	multiplug	26,927
6	sality	23,278
7	zango	21,910
8	solimba	21,305
9	ibryte	20,058
10	expiro	16,685

(c) Andrubis.

#	Label	Samples
1	opfake	88,723
2	fakeinst	84,485
3	smsagent	24,121
4	plankton	22,329
5	kuguo	19,497
6	smsreg	15,965
7	waps	12,055
8	utchi	7,949
9	droidkungfu	7,675
10	ginmaster	6,365

Table 9: University dataset clustering with 4, 10, 48, and all AVs.

AVs	Clusters	Singletons	Unlabeled	Largest
All	1,465,901	1,456,375	1,394,168 (19.2%)	vobfus (701,775)
48	1,543,510	1,534,483	1,472,406 (20.3%)	vobfus (701,719)
10	3,732,626	3,730,304	3,728,945 (51.4%)	multiplug (637,787)
4	5,655,991	5,655,243	5,654,819 (77.9%)	vobfus (539,306)

are singletons. However, these only represent 1.4%–20% of the samples. Thus, the vast majority of samples are grouped with other samples. Singleton clusters can be samples for which no label can be extracted as well as samples assigned a label not seen in other samples. Overall, the percentage of unlabeled samples varies from 1.3% (Andrubis) up to 19.2% (University). All AV labels for these samples are generic and AVCLASS could not identify a family name in them.

Table 8 presents the top 10 clusters in the 3 largest unlabeled datasets (University, Miller, Andrubis). The most common family in both Windows datasets is *vobfus*. Top families in these two datasets are well known except perhaps *flooder*, which the author building the manual lists thought it was generic, but the automatic generic token detection does not identify as such. This is an example of tokens that may sound generic to an analyst, but may be consistently used by AV vendors for the same family. In the University dataset 6 of the top 10 families are malware (vobfus, multiplug, virut, toggle, flooder, upatre) and 4 are PUP (softpulse, loadmoney, zango, ibryte). In the Miller dataset 3 are malware (vobfus, zbot, sality) and 7 PUP (firseria, installerex, domaiq, installcore, loadmoney, hotbar, ibryte). This matches observations in Malsign [15] that large "malware" datasets actually do contain significant amounts of PUP. The Andrubis top 10 contains 4 families that also sound generic (opfake, fakeinst, smsagent, smsreg). However, these families are included as such in the ground truth of the labeled Android datasets (MalGenome, Drebin). While these labels may be used specifically for a family, we believe AV vendors should try choosing more specific family names to avoid one vendor using a label for a family and another using it generically for a class of malware.

Number of AV vendors used. To evaluate the effect of using an increasing number of AV vendors into the labeling process, we repeat the clustering of the University dataset using the same fixed sets of AV vendors used in some prior work: VAMO (4 vendors), Drebin (10), and AV-meter (48). The results in Table 9 show that increasing the number of AV vendors reduces the fraction of samples for which a label cannot be obtained (Unlabeled column). This motivates the design choice of AVCLASS to include AV labels from any available vendor.

5 Discussion

This section discusses AVCLASS limitations, usage, and areas for future work.

As good as the AV labels are. AVCLASS extracts family information AV vendors place in their labels, despite noise in those labels. But, it cannot identify families not in the labels. More specifically, it cannot label samples if at least 2 AV engines do not agree on a non-generic family name. Results on our largest unlabeled dataset show that AVCLASS cannot label 19% of the samples, typically because those labels only contain generic tokens. Thus, AVCLASS is not a panacea for malware labeling. If AV vendors do not have a name for the sample, it cannot be named.

Clustering accuracy. AVCLASS is a malware labeling tool. While it can be used for malware clustering, its evaluated precision is 87.2%–95.3%. This is below state-of-the-art malware clustering tools using static and dynamic features, which can reach 98%–99% precision. As shown in Appendix Table 10, when comparing F1 measure, tools like Malheur [29] (F1= 95%), BitShred [13] (F1=93.2%), and FIRMA [26] (F1=98.8%) outperform AVCLASS. Thus, AVCLASS should only be used for clustering when a state-of-the-art clustering system is not available and implementing one is not worth the effort (despite improved accuracy).

Building reference datasets. When using AVCLASS to build reference datasets, there will be a fraction of samples (up to 19% in our evaluation) for which AVCLASS cannot extract a label and others for which the confidence (i.e., number of AV engines using the chosen family name) is low. While those can be removed from the reference dataset, this introduces selection bias by removing the harder to label samples [16].

AV label granularity. Our evaluation shows that AV labels are not granular enough to differentiate family versions, e.g., DroidKungFu1 from DroidKungFu2. Thus, when releasing labeled datasets, researchers should clearly differentiate the family name from the family version (if available), enabling users to decide which granularity to use.

Validation with real ground truth. To evaluate AVCLASS, we have assumed that the labels of publicly available datasets are perfectly accurate and have compared accuracy to those. However, those labels may contain inaccuracies, which would affect our results either positively or negatively. This can only be resolved by using real ground truth datasets. How to obtain such real ground truth is an important area for future work.

Generic token detection. Our generic token detection requires labeled samples. This creates a bit of a chicken-and-egg problem, which we resolve by using publicly avail-

able labeled datasets. We also release a file with the generic tokens we identified so that users can skip this step. We leave the development of techniques to identify generic tokens that do not require ground truth for future work.

6 Conclusion

In this work we have described AVCLASS, an automatic labeling tool that given the AV labels for a potentially massive number of malware samples, outputs the most likely family names for each sample. AVCLASS implements novel techniques to address 3 key challenges: normalization, removal of generic tokens, and alias detection.

We have evaluated AVCLASS over 10 datasets, comprising 8.9 M samples, larger than any previous dataset used for malware clustering or classification. The results show that the fully automated approach used by AVCLASS can achieve clustering accuracy between 93.9 and 62.3 depending on the dataset. We have compared the generic token and aliases lists automatically produced by AVCLASS with the manual ones produced by an analysts observing that the achieve comparable accuracy in most datasets. We have shown that an increasing number of AV vendors reduces the percentage of samples for which a (non-generic) family name cannot be extracted, thus validating the design choice of using all AV engines. We have also observed that AV labels are not fine-grained enough to distinguish different versions of the same family.

Finally, we have released AVCLASS's source code to the community, along with precompiled lists of alias and generic tokens.

7 Acknowledgments

We specially thank Manos Antonakakis and Martina Lindorfer for providing us with the University and Andrubis datasets, respectively. We also thank the authors of the Drebin, MalGenome, Malheur, Malicia, and the Malicious Content Detection Platform datasets for making them publicly available. We are grateful to Srdjan Matic for his assistance with the plots, Davide Balzarotti and Chaz Lever for useful discussions, VirusTotal for their support, and Pavel Laskov for his help to improve this manuscript.

This research was partially supported by the Regional Government of Madrid through the N-GREENS Software-CM project S2013/ICE-2731 and by the Spanish Government through the Dedetis Grant TIN2015-7013-R. All opinions, findings and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

References

1. D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck. Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. In *Network and Distributed System Security*, 2014.

2. M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *International Symposium on Recent Advances in Intrusion Detection*, 2007.
3. U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *Network and Distributed System Security*, 2009.
4. D. Beck and J. Connolly. The Common Malware Enumeration Initiative. In *Virus Bulletin Conference*, 2006.
5. P.-M. Bureau and D. Harley. A dose by any other name. In *Virus Bulletin Conference*, 2008.
6. J. Canto, M. Dacier, E. Kirda, and C. Leita. Large Scale Malware Collection: Lessons Learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems*, 2008.
7. CARO Virus Naming Convention. http://www.caro.org/articles/naming.html.
8. G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-Scale Malware Classification using Random Projections and Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.
9. I. Gashi, B. Sobesto, S. Mason, V. Stankovic, and M. Cukier. A Study of the Relationship between Antivirus Regressions and Label Changes. In *International Symposium on Software Reliability Engineering*, 2013.
10. D. Harley. The Game of the Name: Malware Naming, Shape Shifters and Sympathetic Magic. In *International Conference on Cybercrime Forensics Education & Training*, 2009.
11. W. Huang and J. W. Stokes. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.
12. M. Hurier, K. Allix, T. Bissyandé, J. Klein, and Y. L. Traon. On the Lack of Consensus in Anti-Virus Decisions: Metrics and Insights on Building Ground Truths of Android Malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.
13. J. Jang, D. Brumley, and S. Venkataraman. BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *ACM Conference on Computer and Communications Security*, 2011.
14. A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. Tygar. Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels. In *ACM Workshop on Artificial Intelligence and Security*, 2015.
15. P. Kotzias, S. Matic, R. Rivera, and J. Caballero. Certified PUP: Abuse in Authenticode Code Signing. In *ACM Conference on Computer and Communication Security*, 2015.
16. P. Li, L. Liu, D. Gao, and M. K. Reiter. On Challenges in Evaluating Malware Clustering. In *International Symposium on Recent Advances in Intrusion Detection*, 2010.
17. M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer. ANDRUBIS-1,000,000 Apps Later: A View on Current Android Malware Behaviors. In *International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, 2014.
18. F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero. Finding Non-Trivial Malware Naming Inconsistencies. In *International Conference on Information Systems Security*, 2011.
19. B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu, A. D. Joseph, and J. D. Tygar. Reviewer Integration and Performance Measurement for Malware Detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.
20. A. Mohaisen and O. Alrawi. AV-Meter: An Evaluation of Antivirus Scans and Labels. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2014.
21. A. Nappa, M. Z. Rafique, and J. Caballero. The MALICIA Dataset: Identification and Analysis of Drive-by Download Operations. *International Journal of Information Security*, 14(1):15–33, February 2015.

22. J. Oberheide, E. Cooke, and F. Jahanian. CloudAV: N-version Antivirus in the Network Cloud. In *USENIX Security Symposium*, 2008.
23. R. Perdisci, A. Lanzi, and W. Lee. McBoost: Boosting Scalability in Malware Collection and Analysis using Statistical Classification of Executables. In *Annual Computer Security Applications Conference*, 2008.
24. R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *USENIX Symposium on Networked Systems Design and Implementation*, 2010.
25. R. Perdisci and U. ManChon. VAMO: Towards a Fully Automated Malware Clustering Validity Analysis. In *Annual Computer Security Applications Conference*, 2012.
26. M. Z. Rafique and J. Caballero. FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In *International Symposium on Research in Attacks, Intrusions and Defenses*, 2013.
27. M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos. CAMP: Content-Agnostic Malware Protection. In *Network and Distributed System Security*, 2013.
28. K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and Classification of Malware Behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2008.
29. K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic Analysis of Malware Behavior using Machine Learning. *Journal of Computer Security*, 19(4), 2011.
30. Virusshare. http://virusshare.com/.
31. Virustotal. https://virustotal.com/.
32. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras. DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications. In *European Symposium on Research in Computer Security*, 2014.
33. Y. Zhou and X. Jiang. Dissecting Android Malware: Characterization and Evolution. In *IEEE Symposium on Security and Privacy*, 2012.

A Additional Results

Table 10: Accuracy numbers reported by prior clustering works.

Work	**Metrics**
Bailey et al. [2]	Consistency=100%
Rieck et al. [28]	Labels prediction=70%
McBoost [23]	Accuracy=87.3%, AUC=0.977.
Bayer et al. [3]	Quality(Prec*Rec)=95.9
Malheur [29]	F1= 95%
BitShred [13]	Prec=94.2%, Rec=92.2%
VAMO [25]	F1=85.1%
Malsign [15]	Prec=98.6%, Rec=33.2%, F1=49.7%
AVCLASS	Prec=95.2%, Rec=92.5%, F1=93.9%

Table 11: AV engines found in our datasets and their lifetime in days.

Engine	First Scan	Last Scan	Days
Ikarus	22/05/2006	29/03/2016	3599
TheHacker	22/05/2006	29/03/2016	3599
F-Prot	22/05/2006	29/03/2016	3599
Fortinet	22/05/2006	29/03/2016	3599
BitDefender	22/05/2006	29/03/2016	3599
CAT-QuickHeal	22/05/2006	29/03/2016	3599
AVG	22/05/2006	29/03/2016	3599
Microsoft	22/05/2006	29/03/2016	3599
ClamAV	22/05/2006	29/03/2016	3599
Avast	22/05/2006	29/03/2016	3599
McAfee	22/05/2006	29/03/2016	3599
TrendMicro	22/05/2006	29/03/2016	3599
VBA32	22/05/2006	29/03/2016	3599
Symantec	22/05/2006	29/03/2016	3599
Kaspersky	22/05/2006	29/03/2016	3599
Panda	22/05/2006	29/03/2016	3599
DrWeb	22/05/2006	29/03/2016	3599
Sophos	22/05/2006	29/03/2016	3599
F-Secure	07/02/2007	29/03/2016	3338
AhnLab-V3	14/03/2007	29/03/2016	3303
Norman	22/05/2006	30/05/2015	3294
Rising	26/07/2007	29/03/2016	3169
AntiVir	22/05/2006	03/09/2014	3025
GData	12/05/2008	29/03/2016	2878
ViRobot	24/07/2008	29/03/2016	2805
K7AntiVirus	01/08/2008	29/03/2016	2797
Comodo	05/12/2008	29/03/2016	2671
nProtect	14/01/2009	29/03/2016	2631
McAfee-GW-Edition	19/03/2009	29/03/2016	2567
Antiy-AVL	24/03/2009	29/03/2016	2562
eSafe	16/11/2006	16/09/2013	2496
Jiangmin	16/06/2009	29/03/2016	2478
VirusBuster	13/06/2006	18/09/2012	2288
eTrust-Vet	22/05/2006	22/05/2012	2191
TrendMicro-HouseCall	04/05/2010	29/03/2016	2156
SUPERAntiSpyware	12/07/2010	29/03/2016	2087
Emsisoft	20/07/2010	29/03/2016	2079
VIPRE	17/11/2010	29/03/2016	1959
PCTools	21/07/2008	23/10/2013	1919
Authentium	22/05/2006	29/04/2011	1803
ByteHero	20/08/2011	29/03/2016	1683
Sunbelt	30/11/2006	29/04/2011	1611
TotalDefense	15/05/2012	29/03/2016	1414
NOD32	24/09/2008	19/07/2012	1394
ESET-NOD32	11/07/2012	29/03/2016	1357
Commtouch	18/01/2011	28/08/2014	1317
Agnitum	18/09/2012	29/03/2016	1288
Kingsoft	18/09/2012	29/03/2016	1288
MicroWorld-eScan	02/10/2012	29/03/2016	1274
NANO-Antivirus	28/11/2012	29/03/2016	1217
Malwarebytes	30/11/2012	29/03/2016	1215
K7GW	15/04/2013	29/03/2016	1079
Prevx	13/05/2009	23/04/2012	1076
NOD32v2	22/05/2006	19/01/2009	973
Ewido	22/05/2006	20/01/2009	973
eTrust-InoculateIT	22/05/2006	15/01/2009	968
UNA	22/05/2006	15/01/2009	968
Baidu	02/09/2013	29/03/2016	939
Baidu-International	03/09/2013	29/03/2016	938
F-Prot4	30/06/2006	15/01/2009	929
Bkav	13/09/2013	29/03/2016	928
Antivir7	22/06/2006	05/01/2009	928
CMC	13/09/2013	29/03/2016	928
T3	14/07/2006	15/01/2009	915
Prevx1	15/11/2006	12/05/2009	909
Ad-Aware	26/11/2013	29/03/2016	854
SAVMail	03/10/2006	18/01/2009	838
Qihoo-360	21/01/2014	29/03/2016	798
AegisLab	29/01/2014	29/03/2016	790
McAfee+Artemis	21/11/2008	18/01/2011	787
PandaBeta	12/02/2007	10/02/2009	729
Zillya	29/04/2014	29/03/2016	700
FileAdvisor	19/02/2007	18/01/2009	699
Tencent	13/05/2014	29/03/2016	686
Zoner	22/05/2014	29/03/2016	677
Cyren	22/05/2014	29/03/2016	677
Avira	22/05/2014	29/03/2016	677
Webwasher-Gateway	20/03/2007	19/01/2009	671
AVware	28/07/2014	29/03/2016	610
a-squared	24/12/2008	28/07/2010	581
Avast5	03/03/2010	28/09/2011	573
McAfeeBeta	04/07/2007	18/01/2009	564
FortinetBeta	01/08/2007	18/01/2009	535
PandaBeta2	07/09/2007	16/01/2009	496
ALYac	26/11/2014	29/03/2016	489
AhnLab	14/03/2007	03/07/2008	477
Alibaba	12/01/2015	29/03/2016	442
NOD32Beta	24/09/2008	16/08/2009	325
Arcabit	02/06/2015	29/03/2016	301
SecureWeb-Gateway	26/09/2008	14/04/2009	200
VIRobot	23/07/2008	17/01/2009	177
Command	17/11/2010	29/04/2011	163
PandaB3	04/09/2008	19/01/2009	136
eScan	25/09/2012	15/10/2012	19
DrWebSE	18/01/2015	03/02/2015	15
ESET NOD32	26/06/2012	26/06/2012	0
Yandex	29/03/2016	29/03/2016	0
TotalDefense2	16/04/2015	16/04/2015	0
SymCloud	11/08/2015	11/08/2015	0