Family Ties: A Close Look at the Influence of Static Features on the Precision of Malware Family Clustering

Antonino Vitale*, Kevin van Liebergen†, Juan Caballero†, Savino Dambra‡,
Platon Kotzias§, Simone Aonzo*, Davide Balzarotti*

*EURECOM
†IMDEA Software Institute
‡Gen Digital
§BforeAI

***Abstract*—Malware family clustering plays a crucial role in many security tasks, including malware analysis, classification, labeling, triage, threat hunting, and lineage studies. This work takes a close look at the influence on malware family clustering of 11 popular static similarity features, including whole-file fuzzy hashes (e.g., SSDeep, TLSH), structural hashes (e.g., PE Hash, Import Hash, VirusTotal's VHash), certificate-based features, and icon-based features. Our goal is not to propose new features or clustering approaches. Instead, we aim to measure how often these 11 features make clustering errors, i.e., cluster together samples belonging to different malware families. We also investigate the root causes behind those errors, which often lead to misinterpretations of malware relationships, hinder effective threat detection, and propagate inaccuracies in downstream analyses. To study this phenomenon, we leverage three public datasets comprising 79,993 labeled Windows malware samples. We cluster those samples by using each of the analyzed features, measure their accuracy with a focus on their precision, and examine the reasons that caused some clusters to contain samples from different families. Our analysis identifies intrinsic limitations of some of the features and highlights the severe impact of EXE-building tools (like software protectors, installers, and self-extracting archives) on malware clustering. Finally, we discuss mitigations and evaluate potential improvements to address the problems we observed. Our findings provide a critical foundation for improving static malware clustering methodologies by emphasizing the importance of dataset curation and feature refinement for robust and precise clustering outcomes.**

I. Introduction

Clustering malicious executables into families is a fundamental problem in malware analysis. It allows studying groups of samples derived from the same malicious source code, which is more efficient than analyzing individual executables. For instance, malware family clustering has been used to aid *classification* [15], to reduce the effort of manual *labeling* [54], to discover new threats [9], for malware *triage* [25], and as a preliminary step to study malware *lineage* [18], [12].

To group malware samples belonging to the same family, clustering approaches may use static features extracted directly from executables, dynamic features extracted from the runtime behavior, or a combination of both. Static features are cheaper to extract and thus are more commonly used in applications that require the analysis of a large numbers of samples. They can also avoid the common mistake of grouping unrelated samples only because they exhibit a similar runtime behavior, e.g., putting together different downloader families in the same cluster.

Different classes of static features exist. For example, binary code similarity approaches [55] extract features from the executable code. However, these approaches require a correct disassembly of the malware code, which is very challenging in presence of obfuscation [31], and may also require complex analysis of control and data flows. This is why most of the existing approaches focus instead on features that can be easily and efficiently extracted, and thus can be applied to very large numbers of samples. Most popular amongst these are whole-file fuzzy hashes like SSDeep [29] and TLSH [43], which produce similar digests for similar input files. Another highly scalable approach is to compute a hash over a subset of an executable such as the import table (*imphash*) [1], selected fields in the PE headers (*pehash*, *richpe*) [61], [60], or the certificate table [30], [26]. We will focus on these *scalable* features in our study.

Researchers have proposed multiple supervised malware classifiers based on machine learning techniques [63], [16], [56], [62], [13]. However, supervised classifiers can only accurately classify samples of families observed during model training. Their performance significantly degrades when applied to out-of-distribution data where samples from new families may appear. In contrast, malware clustering approaches can handle previously unknown malware families by capturing similarity between samples. An accurate malware clustering approach should optimize both *precision* and *recall*, or the F1-score that combines both. However, a key observation about

979-8-3315-8969-1/25/$31.00 © 2025 European Union

malware clustering is that, while *recall* is an important metric (i.e., we do not want samples of a given family to be subdivided in many clusters), *precision* is always paramount (i.e., it is rarely acceptable when samples of different families are erroneously combined within the same cluster). In fact, while a lower recall might result in more manual work for the analysts (e.g., to label the different clusters), a poor precision often leads to wrong results and conclusions. Since precision is instrumental in identifying similarities between different families, while recall is more focused on uncovering diversity within samples from the same family, the primary objective of this paper is to investigate the former aspect. In fact, despite a vast amount of research available on malware clustering, little is still known about the practical limitations that affect the precision of popular static features. In particular, both *when* they fail and, most importantly, *why* they fail are two aspects still poorly understood.

To cover this gap, in this paper we examine the limitations of the precision of a number of widely-used, highly scalable, static similarity features. Our emphasis is on understanding the underlying causes of errors that result in the misclassification of samples from different malware families that are incorrectly placed in the same cluster.

It is important to emphasize that our aim is not to propose novel malware clustering approaches, nor to critically evaluate or challenge existing ones. On the contrary, we contend that the insights derived from this investigation will offer valuable contributions to the broader field of malware clustering research. Specifically, for each static feature analyzed, we explore the factors that lead to the erroneous grouping of samples from distinct families, thereby reducing the overall precision.

To this end, we leverage three public datasets of malicious Windows executables labeled with their family [13], [23], [40]. We cluster the 79,993 unique samples using separately each of 11 popular static similarity features. We selected these features for their proven ability to effectively cluster malware samples, utilizing established methods from both industry and academic research. These features are commonly available through online malware analysis services, such as VirusTotal [57], and have been used in previous studies for clustering based on static features [14], [43], [34], [30], [26], [27], [6], [42], [10], [64]. Then, we compute the clustering accuracy[1] obtained by each feature individually, with an emphasis on precision. Finally, we examine the *mixed clusters* containing samples from different families according to the ground truth labels. For this examination, we leverage 8 analysis features that capture possible reasons why samples from different families may be grouped together. For instance, these include being generated by EXE-building tools (e.g., packers, installers, compressors), sharing common overlay data, or being erroneously truncated. Using this approach, we answer two research questions.

RQ1: What is the precision of the most commonly-adopted static features when used for malware family clustering? Our analysis identifies three groups of features according to their precision. The first group contains three features related to code signing with nearly perfect precision (over 99%). However, their ability to group samples is limited, making them especially suitable for verifying the correctness of ground truth labels during dataset construction. The second group achieves high precision (in the range of 94% to 97%) and comprises structural hashes such as *pehash* and VirusTotal's proprietary *vhash*, and whole-file fuzzy hashes such as *SSDeep* and *TLSH*. In this group, *pehash* is particularly effective, offering the highest precision and average cluster size. The third group contains features with lower precision, below 90%, and includes *imphash*, *richpe*, and icon-based features. These features exhibit frequent collisions in their values, thus introducing significant errors in identifying samples from the same family.

RQ2: What are the main limits of those features for malware family clustering? When should analysts be careful with their use? The main limits of the analyzed similarity features stem from their sensitivity to EXE-building tools, which often introduce similarities unrelated to malware family characteristics. The impact of EXE-building tools is small in the first group of features (high precision), but significant for the other two groups where the largest clusters are mixed, i.e., contain samples from different families. Our experiments show that for fuzzy hashes like *SSDeep* and *TLSH*, up to 45% of the mixed clusters can be caused by EXE-building tools. Beyond EXE-building tools, features in the third group (lowest precision) also suffer from inherent weaknesses, such as collisions caused by short import and rich header tables, and the use of generic icons. We also identify cases where the precision is lowered, not because of feature limitations, but due to erroneous ground truth labels.

Finally, we evaluate whether pre-processing techniques such as avoiding to group samples known to be generated by EXE-building tools, with invalid certificate chains, or with short import tables can solve the issues. The precision increase is highest for features in the third group, especially for *imphash*, where precision improves by 10.3%. However, even with pre-processing, some limitations persist, such as those related to generic icons.

In conclusion, the findings of our study shed light on numerous limitations of popular static features for malware family clustering, emphasize the importance of careful feature selection, and the role of preprocessing to mitigate errors and maximize the precision.

II. Motivation and Background

This section presents background on malware clustering based on static features, reviewing commonly used similarity measures and motivating our focus on understanding the factors that negatively impact their clustering precision.

Extensive research has addressed the problem of clustering malware samples into families [9], [45], [22], [20], [48], [34], [42]. The most closely related to this paper are previous works

[1]Henceforth, we use the term *accuracy* as an overarching designation for measures of predictive performance, rather than the *Accuracy* metric itself.

TABLE I
RELATED WORK ON CLUSTERING USING STATIC FEATURES.

Work	Year	Authentihash	Cert Subject	Cert Thumbprint	Icon DHash	Icon Hash	Imphash	PEHash	RichPE	ssdeep	TLSH	vhash
PEHash [61]	2009	-	-	-	-	-	-	✓	-	-	-	-
French et al. [14]	2012	-	-	-	-	-	-	-	-	✓	-	-
TLSH [43]	2013	-	-	-	-	-	-	-	-	-	✓	-
Certified PUP [30]	2015	✓	✓	-	-	-	-	✓	-	-	-	-
Li et al. [34]	2015	-	-	-	-	-	-	-	-	✓	-	-
Webster et al. [60]	2017	-	-	-	-	-	-	-	✓	-	-	-
Certified Malware [26]	2017	-	✓	✓	-	-	-	-	✓	-	-	-
Chikapa et al. [11]	2018	-	-	-	-	-	✓	✓	-	-	-	-
Joyce et al. [24]	2019	-	-	-	-	-	✓	✓	✓	-	-	-
Posluvsny et al. [47]	2019	-	-	-	-	-	-	-	✓	-	-	-
Kim et al. [27]	2020	-	-	-	✓	-	-	-	-	-	-	-
Fuzzy-Import [38]	2020	-	-	-	-	-	✓	-	-	✓	-	-
Namanya et al. [39]	2020	-	-	-	-	-	✓	✓	-	✓	-	-
Ali et al. [6]	2020	-	-	-	-	-	-	-	-	-	✓	-
HAC-T [42]	2020	-	-	-	-	-	-	-	-	-	✓	-
Botacin et al. [10]	2021	-	-	-	-	-	-	-	-	✓	-	-
RecMaL [64]	2023	✓	-	-	-	-	✓	✓	-	-	-	✓
VirusTotal [57]	2024	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓

that use static features for malware family clustering, often evaluating their ability to group similar samples or improve detection rates. Table I captures works that have used at least one of the similarity features analyzed in this paper, and provides a broad view of the features that have received the most attention.

Among these approaches, only two studies [30], [26] consider the use of certificates for grouping samples belonging to the same entity. Additionally, only one study [64] evaluates the accuracy of Authentihash grouping, irrespective of whether the samples are signed or not.

Kim et al. [27] evaluated image similarity algorithms for malware icons. Other crypto-hash features, namely Imphash [1], PEHash [61], RichPE [60], and vhash [58], are designed to cluster samples sharing common structures. Approaches based on these features can be divided into two main groups: those that evaluate the overall precision of individual features [11], [24], [47], and those that combine a subset of them to improve clustering quality [64], [30], [26]. While these studies assess the clustering performance of the features, they do not provide a detailed explanation of the factors influencing their effectiveness.

Like crypto-hash features, fuzzy hashes are designed to group similar variants. Many studies evaluate their precision against different datasets [14], [43], [34], [6], [42], [10], [44]. Naik et al. [38] proposed combining SSDeep and Import Hash to increase malware detection rates. Similarly, Namanya et al. [39] combined PEHash, Import Hash, and SSDeep over the resource section only and over the whole file, achieving the best precision and detection rates. These approaches demonstrate how combinations of features can yield better clustering outcomes, but they often treat the features as black boxes, without analyzing what content drives clustering errors.

To summarize, most of the works incorporate clustering features such as PEHash [61], [30], [11], [24], [39], [64], Import Hash [11], [24], [38], [39], [64], RichPE [60], [26], [24], [47], and SSDeep [14], [34], [38], [39], [10] to evaluate or enhance clustering accuracy. Conversely, some features are rarely considered, e.g., certificate-related [30], [26], [64], Icon DHash [27], TLSH [43], [6], [42], and vhash [64], while Icon Hash never is.

TABLE II
DATASETS USED AND IMBALANCE METRICS: COEFFICIENT OF VARIATION (CV) AND KULLBACK-LEIBLER DIVERGENCE (KL). LOWER MEANS MORE BALANCED.

Dataset	Samples	Families	Collection	CV	KL
MalPE [13]	67,000	670	08/2021 - 03/2022	0.0	0.0
MOTIF [23]	3,095	454	01/2016 - 12/2020	1.69	0.71
Malicia [40]	9,908	53	03/2012 - 02/2013	4.56	2.71
Superset	79,993	1,177	03/2012 - 03/2022	2.77	0.72

Despite these efforts, existing works primarily focus on measuring the effectiveness of static features in clustering or classification tasks, without investigating the underlying causes of clustering failures. For instance, Dambra et al. [13] identify which static features are most influential in popular machine learning models for malware classification, but do not explore why these features succeed or fail in clustering scenarios. Other studies, such as Anderson et al. [7] and Mohaisen et al. [36], also analyze static feature sets in the context of classification or detection, but treat feature outputs as black boxes and do not provide an explanation of misgrouping behavior. In contrast, our work aims to fill this gap by systematically analyzing not only how well each feature performs, but also why it may fail, providing insights into the structural and semantic properties that mislead clustering algorithms.

III. DATASETS

In their seminal work on the evaluation of malware clustering, Li et al. [32] showed that the approach used to assemble ground truth (GT) data and the balance between different families introduced a considerable bias in the accuracy of previous clustering experiments, often leading to vastly different results when the same technique was used on different malware datasets. To mitigate these problems, we use three datasets of malware samples, as summarized in Table II. All three datasets consist of Windows PE executables labeled with the family to which the sample belongs. Family labels were obtained differently in each dataset, thus introducing diversity in the GT methodology. Moreover, the families in the datasets belong to a variety of malware classes, such as adware, backdoors, downloaders, ransomware, rogueware, viruses, and worms, ensuring a diverse and representative sample set for our analysis.

MOTIF. The Malware Open-source Threat Intelligence Family (MOTIF) dataset contains 3,095 PE malware samples from 454 families [23]. Samples and family labels come from open-source threat reports published by 14 major cybersecurity organizations over a period of five years, between January 2016 and December 2020. We downloaded the list of sample hashes and family names from the MOTIF repository [4] and then retrieved the samples from VT. In this dataset, samples are not evenly distributed among families. Of the 454 families, 131 (29%) have only one sample, 232 (51%) contains between two and nine samples, and 91 (20%) have ten or more samples. Only one family (*icedid*) is associated to more than 100 samples. Whenever a sample is assigned a group of labels (considered aliases), we used AVClass [50] to select the most representative name, or simply assign the most popular label in case AVClass is not able to disambiguate.

Malicia. The Malicia dataset contains 9,908 Windows PE malware samples collected from drive-by downloads between March 2012 and February 2013 [40]. Labels for the samples were generated by clustering the samples using network features, screenshots obtained during the sample's execution, and the embedded icon (if present). The 53 clusters were then manually labeled by human experts, resulting in 19 well-known family names and 34 cluster identifiers assigned when family names were not known. Also in this case, samples are not evenly distributed. The dataset contains 23 (43%) families with only one sample, 17 (32%) families with up to nine samples, and 13 (25%) families with at least 10 samples. Only 4 families have at least 100 samples and the largest family (*winwebsec*) is associated to a stunning 5,820 samples, which constitutes more than half (58.7%) of all samples in the dataset. Although the distribution of the Malicia dataset has been discontinued, we contacted the original authors, who kindly granted us access to the dataset for research purposes.

MalPE. Since both Malicia and MOTIF are relatively small and largely unbalanced datasets, we also added the recent *balanced malware* dataset assembled by Dambra et al. [13] (hereinafter referred to as MalPE). The dataset contains 67,000 hashes of 32-bit PE files that appeared in the VirusTotal (VT) feed between August 2021 and March 2022. The samples are equally divided among 670 malware families with 100 samples per family. The authors obtained the family names by using the AVClass tool. We obtained the list with the hashes and family labels from the authors' GitHub repository [3] and then downloaded each sample from VT.

VT reports. We collected reports on all samples through the VirusTotal (VT) API in October 2024. All samples were already known to VT as dataset authors had submitted their samples in the past. We use the VT reports to extract features for the samples, such as the sample hashes (MD5, SHA1, SHA256), scan results with AV engines, and the date the sample was first submitted to VT.

Superset. The datasets use different hashes to identify the samples, e.g., MalPE uses SHA256, Malicia uses MD5, and MOTIF provides multiple hashes. We use the VT reports to get the SHA256 hashes for each sample and use them to uniquely identify samples. We identify 79,993 unique samples across the 3 datasets, tagged with 1,177 family names. These 79,993 samples comprise our *Superset* dataset. At the sample level, the three datasets are largely disjoint. Only eight samples appear both in MalPE and MOTIF and two samples were common between MalPE and Malicia. There are no common samples between MOTIF and Malicia. Of the eight samples shared by MalPE and MOTIF, five were assigned the same family name in both datasets, while three had different families. One sample is assigned *disttrack* in MalPE and *shamoon* in MOTIF, which are aliases according to AVClass. Of the two samples in common between MalPE and Malicia, one was assigned the same family name in both datasets and the other had different ones.

Thus, in total we found three samples with incompatible family names. To avoid inconsistencies in labeling, we removed the overlapping samples from Malicia and MOTIF, retaining them only in the MalPE dataset when building the Superset. When analyzing one dataset, we include all dataset samples. When analyzing the Superset dataset, we consider the 79,993 unique samples. One family (*ramnit*) appears in all three datasets and 51 families appear in two datasets. The largest overlap is between MOTIF and MalPE with 46 families in common. MalPE and Malicia share 6 families and MOTIF and Malicia two.

IV. Features

After a careful review of the static features used in the literature, detailed in Section II, we selected 11 features for clustering the samples (described in Section IV-A), 8 features for analyzing the clustering results (described in Section IV-B), and the AVClass [50] labeling results as a baseline for comparison.

A. Similarity Features

The top part of Table III summarizes the 11 similarity features used to cluster samples in the datasets. The list includes six crypto hashes, calculated over different parts of a PE file: *imphash* covers the import table [1]; *pehash* covers selected fields in the PE headers [61]; *richpe* covers the optional Rich Header that contains information about the compilation of modules in the PE executable [60]; *cert_thumbprint* covers the leaf certificate for signed samples; *authentihash* covers the full

TABLE III
FEATURES USED AND PRESENCE IN SUPERSET DATASET. ON TOP THE 11 SIMILARITY FEATURES USED FOR CLUSTERING SAMPLES, FOLLOWED BY AVCLASS, AND BELOW THE 8 ANALYSIS FEATURES USED TO ANALYZE CLUSTERING RESULTS.

Feature	Type	Source	Samples	Values	Clustering
authentihash	cryptohash	PE	79,993 (100.0%)	77,222	FVG
cert_subject	string	VT	8,906 (11.13%)	1,610	FVG
cert_thumbprint	cryptohash	VT	8,906 (11.13%)	2,004	FVG
icon_dhash	fuzzyhash	VT	45,820 (57.28%)	9,138	FVG
icon_hash	cryptohash	VT	45,820 (57.28%)	15,166	FVG
imphash	cryptohash	PE	77,245 (96.56%)	20,023	FVG
pehash	cryptohash	PE	79,993 (100.0%)	31,367	FVG
richpe	cryptohash	PE	49,815 (62.27%)	11,651	FVG
tlsh	fuzzyhash	PE	79,993 (100.0%)	79,236	HAC-T
ssdeep	fuzzyhash	PE	79,993 (100.0%)	75,535	Single linkage
vhash	structhash	VT	79,823 (99.79%)	27,446	FVG
avclass	string	VT	78,067 (97.59%)	1,270	FVG
die_packer	string list	PE	16,520 (20.65%)	190	-
die_installer	string	PE	3,359 (4.20%)	24	-
die_archive	string list	PE	5,015 (6.27%)	15	-
die_script	string list	PE	2,227 (2.78%)	7	-
packgenome	string list	PE	25,961 (32.45%)	100	-
truncated	Boolean	PE	2,520 (3.15%)	2	-
overlay_sha256	cryptohash	PE	35,461 (36.27%)	30,690	-
no_overlay_sha256	cryptohash	PE	79,993 (100.0%)	61,366	-

PE executable excluding code signing fields (e.g., checksum and Certificate Table) and overlays; and *icon_hash* covers the optional icon image. All cryptographic hashes are compared using equality, i.e., two files have the same hash or not.

The list also includes three fuzzy hashes, which produce similar values for inputs that share some similarities. We use two fuzzy hashes computed over the whole file: *tlsh* and *ssdeep*. *tlsh* uses Hamming distance on the digests to determine if two inputs are similar while *ssdeep* uses Levenshtein edit distance. We also include *dhash*, a perceptual hash applied over the embedded icon image. Perceptual hashes are a special type of fuzzy hashes that produce similar digests for images that look similar to a human (e.g., resized, color changes). We consider two images to match only if their *dhash* digests are identical, effectively grouping together only those that are nearly identical in appearance. It is possible to use Hamming distance with a threshold (e.g., 2 bits) on the *dhash* digest to identify more dissimilar icons, but that reduces the precision.

Another type of hash included is *vhash*, VT's proprietary structural hash. According to VT's minimal documentation on it [58], this hash takes into account properties such as imports, exports, sections, and file size. VT provides daily results of files clustered by *vhash*. Samples in the same cluster always have the same *vhash* value, so we use equality to compare two *vhashes*.

Finally, our list of features includes the leaf certificate subject distinguished name (*cert_subject*), which is useful for identifying signed samples that use different certificates for the same entity.

Most features can be extracted directly from the PE executables, except VT's proprietary *vhash*. However, as shown in the *Source* column in Table III, we also extract the certificate and icon features from the VT reports. The reason for this is that different ways exist for extracting these pieces of information from an executable, and it is therefore more convenient for users to leverage the VT reports to avoid implementing their own method. For example, signed samples contain an unordered sequence of certificates and identifying the leaf certificate requires carefully ordering them. Similarly, an executable may include multiple resources of type icon, and VT selects among those the one it considers the main icon.

As shown in Table III, some features are optional and may not exist in all samples. For example, only 11% of samples are signed, 57% have icons, 62% have a Rich Header, and 96% have an import table.

AVClass. We also consider the malware family obtained by feeding the VT reports to the AVClass malware labeling tool [50], [51]. While the 11 similarity features can be used for clustering similar samples, they do not provide a family name to the clusters. In contrast, AVClass is mainly a labeling tool, that aims to assign a family name to each input sample. However, its output can also be used for clustering, by grouping together samples assigned the same family name. In contrast to the similarity features, we do not know exactly how the AVClass family was generated since it is obtained from a majority voting between the labels assigned by different AV engines, each using their own proprietary techniques (e.g., signatures, machine learning). We use AVClass families as a baseline to compare the clusters obtained by individual similarity features.

B. Analysis Features

The bottom part of Table IV summarizes the 8 analysis features that we use to examine the clustering results. These features capture potential reasons for two different programs to share some binary similarity. We will study how these analysis features correlate with the clusters created using the similarity features.

EXE building tools. Beyond compilers and linkers, there exist other off-the-shelf tools that malware authors can use to build their malicious executables. We consider four classes of such tools: software protectors, installers, self-extracting archives, and tools that embed scripts and their interpreter in an executable. Protectors are used by malware authors for evasive purposes [53], [35]. For instance, protectors may compress or encrypt the code and data of the original program and uncompress or decrypt them at runtime. They may also inspect the runtime environment to identify the presence of analysis tools or security products, refraining from running the malicious behavior in case any are detected. For simplicity, in this work, we refer to software protectors as *packers*. Popular packers include UPX, PECompact, ASPack, and Themida. *Installers* are responsible for performing all installation steps for a target program. They are used when a program requires multiple files beyond the main executable or when the installation requires actions such as creating folders or setting up registry keys. Popular installer builders include InnoSetup, NSIS, InstallShield, WIX, and InstallMate. *Self-extracting (SFX) archives* embed a compressed archive and the decompression routine required to automatically extract the archive when the executable is run. Most popular compressors (e.g., 7-zip, WinRAR, and WinZip) can build SFX archives. Finally, executables can be generated from scripts by embedding the script's code with its corresponding interpreter. Popular tools in this category include aut2exe for AutoIt and bat2exe for BAT.

To identify samples built using the above classes of tools, we leverage the signatures provided by Detect-it-Easy (DiE) [19]. To identify packed samples we also leverage YARA rules for 20 packers built with PackGenome [33]. Other packing detection tools exist (e.g., PEiD [2]), but DiE and PackGenome are the most up-to-date (e.g., PEiD [2] has not been updated in 7 years).

According to DiE, one-third of the samples in the Superset dataset have been produced by EXE-generating tools: 16,520 (20.6%) samples packed with well-known packers, 5,015 (6.2%) SFX archives, 3,359 (4.2%) installers, and 2,227 (2.7%) samples generated from scripts. PackGenome detects 32.4% packed samples.

Overlay. A PE executable has an overlay if it contains additional data appended to its end, whose presence is not revealed by the PE header fields. Overlays can be detected because the size of the file on disk is larger than its expected size (i.e., the sum of the start offset and size of the last section). A stunning 36.2% (35,461) of our samples include an overlay. For each of them, we computed the hash of the overlay's content (*overlay_sha256*) and the hash of the executable without the overlay (*no_overlay_sha256*).

Truncation. Samples may get truncated as part of the malware collection process. From a security analysis perspective, truncated samples can be considered corrupted and cannot be executed. Truncated samples can be identified because their expected size (i.e., the sum of the start offset and size of the last section) is larger than the real size of the file on disk. We identify 2,521 (3.1%) truncated samples: 2,486 (3.7%) in MalPE, 24 (0.8%) in MOTIF, and 11 (0.1%) in Malicia. Although it is especially prevalent in MalPE (and thus in the VT file feed), this confirms that truncation is a widespread phenomenon that affects all datasets. Truncated samples should arguably not be included in malware datasets, as they do not correspond to fully functional programs.

V. Analysis Approach

This section summarizes our approach for analyzing the limits of the similarity features. It comprises three steps: clustering the samples, computing the clustering accuracy, and analyzing the clusters with samples from multiple families.

Clustering. We perform a separate clustering of the samples in each dataset by using each similarity feature in isolation. Depending on the feature, we employ two different clustering approaches. For 10 of the similarity features, we perform a simple feature value grouping (FVG) which places samples with the same feature value in the same cluster, i.e., one cluster per feature value. This approach is used for cryptographic hashes, structural hashes, strings, and boolean features. We also use FVG for the *icon_dhash* fuzzy hash to only group samples with nearly identical icons. In the event that a sample lacks a particular feature, the FVG clustering places the sample in a singleton cluster on its own.

For the other two fuzzy hashes, we employ an agglomerative clustering approach. For *tlsh*, we use HAC-T, a hierarchical agglomerative clustering technique specially designed for TLSH digests [42]. The main advantage of HAC-T is that it is more efficient than traditional hierarchical clustering, which has a quadratic complexity; in contrast, HAC-T has a time complexity of $\mathcal{O}(n \log n)$. For our experiments, we use the suggested clustering hyper-parameter value, i.e., $C_{dist} = 30$ [42]. For *ssdeep*, we rely instead on the built-in single-linkage clustering provided by its official implementation [52]. In this case, clusters are formed based on the presence of at least one pairwise connection above a threshold t. We choose a conservative threshold $t = 70$ since this value appears to provide the best balance between sensitivity and specificity for malware classification [37], [38].

Clustering accuracy. We compute the clustering accuracy using an external clustering validation approach that compares the clustering results with the reference clustering provided by the GT. The external validation evaluates if the two sets of clusters group samples in a similar way. It is important to note that only the structure of the clusters is compared, i.e., the family names in the reference clusters are not used in the evaluation. For each experiment, we report precision, recall, and F1 score, as computed in prior malware clustering works [9], [49], [40], [46]. A clustering has perfect precision if every cluster is *pure*, i.e., contains only samples from a single family. In other words, there are no *mixed clusters* (MCs) containing samples from multiple families. A high recall means that most samples of each family belong to a single cluster, while a low recall indicates that a family is fragmented over multiple clusters.

We examine how often each similarity feature makes errors that reduce its precision, i.e., link samples in different families. We focus on precision because features with high precision can be combined into more sophisticated clustering approaches with minimal errors. Also because when clusters have perfect precision, i.e., contain only samples of one family, then an analyst can simply label one of the samples in the cluster and propagate the label to other samples. While we also provide recall and F1 score metrics, it is important to stress that our goal is not to examine *how good* each individual feature is at clustering samples, as real-world malware clustering approaches would most likely combine multiple features to achieve better results. We focus instead on analyzing the limits of each feature and the reasons they might produce mixed clusters.

Mixed cluster analysis. None of the features has perfect precision, i.e., they all incorrectly group together some samples belonging to different families. To identify the causes behind the errors, we analyze the mixed clusters (MCs), which contain samples from different families and are responsible for lowering the overall clustering precision. For each MC, we compute a distribution of each of the 8 analysis features. For example, we count how many samples are packed with each specific packer identified by DiE or PackGenome, how many samples use each specific installer software, and how many samples are truncated. If any of the analysis features affects all samples in the cluster, then we conclude that the analysis feature offers a *possible explanation* for the MC. For example, if a cluster has 100 samples and DiE identifies that all 100 samples are generated by the InnoSetup installer, we consider that samples in the cluster may have been grouped together (despite belonging to different families) due to being generated using that installer software.

It is worth noting that this approach captures correlation rather than causality. However, while we cannot claim that a given analysis feature necessarily is the cause behind the errors, our results often point to a possible causal link. For instance, the use of a given installer tool does in fact introduce common parts in the resulting program binaries, and this similarity is picked up by some of the features and thus results in erroneously combining otherwise different samples in the same cluster. In any case, in order to gain further insight into the underlying causes of the errors introduced by each feature, we complement the correlation analysis with a manual investigation of the clusters.

VI. Analysis

This section first presents the clustering results in Section VI-A and then analyzes the limits of the similarity features in Section VI-B.

A. RQ1 – Similarity Feature Precision

Table IV summarizes the clustering accuracy of each feature over the Superset dataset. While all individual features provide reasonably high precision, we can clearly identify three groups: three features (*authentihash*, *cert_thumbprint*, *cert_subject*) have a precision above 99%, four features (*pehash*, *ssdeep*, *tlsh*, *vhash*) have a precision between 94% and 97%, and four features (*imphash*, *richpe*, *icon_hash*, *icon_dhash*) have a precision below 90%.

On the other hand, individual features fail to group many samples together: the largest cluster contains 1,651 samples (*icon_dhash*) but the average cluster size (excluding singletons) is always lower than 14.6 samples (icon_dhash). The trend of groups we observed for the precision is reverted, e.g., the group of features with higher precision has the lowest recall. For instance, the feature that is most effective at creating large clusters (*icon_dhash*) is also the one with the lowest precision, i.e., the one that makes more errors.

The comparison with the AVClass labeling tool used as baseline shows that individual similarity features have precision in the same range as a popular malware labeling tool, but the recall (and thus the F1 score) is much lower. This makes sense as malware clustering approaches typically combine multiple features to increase the overall accuracy [8], [9], [45].

B. RQ2 – Similarity Feature Limits

Table V reports the number of MCs obtained with each *similarity feature*, along with the number of such clusters in which all samples share the same *analysis feature*. For instance, a value of 2 in the *Packer* column means that two of the MCs contained all their samples packed with the same packer. Next, we examine the reasons behind the MCs for each feature in order to identify their limits for family clustering.

Authentihash. This feature exhibits the highest precision (0.997) because samples with the same *authentihash* are nearly identical and can only differ in the checksum field, certificate table, and overlay (if these parts exist). On the other hand, it also groups the least number of samples with over 99% of the clusters containing only one sample, and the remaining containing 6.5 samples on average. There are only 7 *authentihash* MCs, all from the MalPE dataset. None of the samples in the MCs include an overlay. Looking at the certificate, two clusters have all samples signed using the same leaf certificate, three have a mixture of signed and unsigned samples but all signed samples use the same leaf certificate, and two have two unsigned samples each. Since the only differences are in the signature-related fields, samples in these MCs have the same code and data, and thus they necessarily belong to the same family. Thus, the reason behind the MCs are instead errors in the GT. In particular, we identify that 4 MCs are due to different MalPE families being in reality aliases. In particular, we identify two alias groups (*ascentive*, *speedcat*, *gamini*, *deceptpcclean*) and (*winwrapper*, *spigot*). We approached the AVClass team about these two groups and they concluded they are indeed aliases that should be incorporated into AVClass. The other MCs are due to three samples being mislabeled in the MalPE GT.

Thus, we conclude that while *authentihash* does not group much, it has nearly perfect precision and therefore we suggest researchers to use it while constructing malware datasets to identify GT errors.

TABLE IV

Clustering results on Superset dataset. For each similarity feature, it shows the number clusters, singleton clusters, non-singleton clusters, largest cluster size, average cluster size excluding singletons, and the accuracy metrics.

	Clusters							
Feature	**All**	**Singl.**	**> 1**	**Max**	**Avg**	**Prec.**	**Recall**	**F1**
authentihash	77,221	76,721	500	185	6.5	0.997	0.026	0.050
cert_subject	72,696	72,003	693	160	11.5	0.991	0.050	0.096
cert_thumbprint	73,090	72,232	858	160	9.0	0.994	0.040	0.077
icon_dhash	43,310	40,618	2,692	1,651	14.6	0.856	0.203	0.328
icon_hash	49,338	46,281	3,057	1,521	11.0	0.888	0.186	0.307
imphash	22,771	17,133	5,638	547	11.1	0.873	0.321	0.470
pehash	31,367	25,342	6,025	345	9.1	0.963	0.230	0.372
richpe	41,524	37,847	3,677	715	11.5	0.887	0.211	0.340
ssdeep	41,367	34,866	6,501	310	6.9	0.967	0.172	0.292
tlsh	40,025	33,412	6,613	342	7.0	0.959	0.174	0.295
vhash	27,615	19,726	7,889	715	7.6	0.939	0.183	0.306
avclass	3,195	2,241	954	2,595	81.5	0.909	0.872	0.890

TABLE V

Number of mixed clusters (MCs) per feature in Superset and possible reasons.

Feature	Mixed	None	Packer	Archive	Script	Inst.	No-Over.	Trunc.
authentihash	7	2	2	1	-	2	-	-
cert_subject	141	113	12	3	1	12	-	-
cert_thumbprint	122	91	13	4	1	13	-	-
icon_dhash	890	764	82	17	11	13	3	-
icon_hash	725	609	75	15	10	12	4	-
imphash	790	434	234	44	35	22	19	2
pehash	418	172	92	36	38	35	38	7
richpe	656	492	93	39	5	16	9	2
ssdeep	443	240	97	41	27	12	23	3
tlsh	504	267	117	40	28	24	22	6
vhash	1,191	387	427	110	88	87	45	47
avclass	361	346	3	1	4	7	-	-
All	6,248	3,917	1,247	351	248	255	163	67

Certificates. The certificate features boast the second and third highest precision with 0.994 for *cert_thumbprint* and 0.991 for *cert_subject*. The latter groups more as it can identify samples with different certificates for the same entity, but at the expense of lower precision. The number of MCs (122–141) is the lowest behind *authentihash*. The most common likely explanations are *packers* and *installers*, which cover 10% of MCs each. Installers are a much more common explanation than other features likely because installers are prevalent among PUPs, and a larger fraction of PUP is signed compared to malware [30]. This is interesting as it shows a clear correlation without causality, as the installer per se does not affect the certificates. Of the 10 largest clusters for *cert_thumbprint*, 6 are pure and the other 4 are MCs due to GT errors already described, thus no real collisions are observed between families in the top 10 clusters. However, we identify a few smaller MCs that are due to invalid certificate chains, including leaf certificates of benign entities such as “Mozilla Corporation”, “Corel Corporation”, and “Opera Software AS”. To avoid such errors we can change the certificate features to only be extracted for properly signed samples. This change increases the precision of both features to 0.996.

We conclude that certificate features have high precision and are rarely affected by EXE-building tools. However, to handle the misuse of benign certificates by different families, we suggest certificate features to be extracted only for properly signed samples.

Whole-file fuzzy hashes. Both SSDeep and TLSH capture whole-file binary-level similarity and achieve similar results, so we discuss them together. Both features achieve high precision (0.967 for SSDeep and 0.959 for TLSH) and limited grouping (6.9 and 7.0 average samples per cluster). SSDeep shows slightly better precision, generating 443 MCs compared to 504 for TLSH. Our analysis features provide a possible explanation for 46%–47% of these MCs, with the most common causes being packing (22%–23%), SFX archives (8%–9%), script-generating tools (5%–6%), overlays (4%–5%), installers (3%–5%), and file truncation (0.7%–1%). Table VI shows the 10 largest SSDeep clusters. Of those, 9 are MCs and 6 are likely caused by EXE-generating tools such as SFX archives (2 MCs), the InnoSetup installer (1), the aut2exe script-building tool (1), and the UPX (1) and dxpack (1) packers. For example, the MC at rank 3 comprises 173 samples from 19 families written in AutoIt. The top families in this MC are *autinject* (70 samples), *autoitinject* (37), and *aitinject* (21), but the cluster also contains families that may wrap samples in AutoIt scripts such as *remcos* [59]. Another two MCs are due to the GT errors already described in the *authentihash* paragraph. The other MC contains 157 samples from two families (*spesr*, *vbinder*) all with an overlay and the exact same content when ignoring the overlay (*no_overlay_sha256*). This behavior is characteristic of prepender viruses that store benign infected program in the file overlay [21]. Interestingly, there is an almost one-to-one mapping with the 10 largest TLSH clusters, the exceptions being the top SSDeep being broken in two with TLSH and the top TLSH cluster broken in three by SSDeep. This suggests that both fuzzy hashes perform similar mistakes and that those mistakes are likely caused by the same underlying reasons.

We conclude that up to 45% of the clusters generated by both SSDeep and TLSH, and most of the largest ones, only capture similarity introduced by EXE-building tools, rather than the similarity of the malicious code and data.

PEhash. This feature achieves high precision (0.963) and less MCs (418) than the whole-file fuzzy hashes. In addition, the use of *PEhash* also tends to group more samples, with an average cluster size of 9.1. The distribution of possible reasons reported in Table V is similar to the one we discussed for fuzzy hashes: packers (19%), script-building tools (9%), SFX archives (9%), overlays (9%), installers (8%), and truncation (1.6%). Out of its 10 largest clusters, 6 are pure and 4 are MCs. The MCs correspond to ranks 1, 2, 4, and 10 in Table VI, two of which are due to SFX archives, one to InnoSetup, and the other to GT errors.

We conclude that *pehash* is slightly better than fuzzy hashes for grouping samples into families, but is similarly affected by EXE-building tools.

Vhash. The proprietary *vhash* has medium precision (0.939), lower than *pehash*, *ssdeep*, and *tlsh*. It groups more samples than *ssdeep* and *tlsh*, but less than *pehash*. The use of this feature results in a stunning 1,191 MC clusters, which correspond to 15% of its non-singleton clusters. This is twice the ratio of other features: *ssdeep* (6.8%), *pehash* (6.9%), and *tlsh* (7.6%). The possible reasons for MCs are dominated by packers (36%). This feature also seems more affected by truncation (4%) than others. Of the largest 10 clusters, 6 are pure and 4 MCs. The 4 MCs correspond to rank 1 (split into two MCs), 2, and 6 in Table VI.

We conclude that *vhash* is worse than *pehash* for grouping PE executables into families, achieving less precision and grouping less. It seems to generate a higher number of collisions too, although its proprietary design makes it difficult to assess the exact reasons.

TABLE VI
OF THE 10 LARGEST SUPERSET CLUSTERS USING SSDEEP, 9 ARE MCS AND 6 ARE LIKELY DUE TO EXE-GENERATING TOOLS.

#	Samples	MC	Families	Likely Reason
1	310	✓	5	installer:inno
2	185	✓	4	GT errors
3	173	✓	19	script:aut2exe
4	167	✓	2	archive:sfx
5	162		1	-
6	157	✓	2	no_overlay_sha256
7	157	✓	2	packer:dxpack
8	149	✓	8	packer:upx
9	132	✓	3	GT errors
10	125	✓	3	archive:sfx

Imphash. This feature has the lowest precision (0.873) and second highest number of MCs (790). On the other hand, it has the highest recall (0.321) and an average cluster size of 11.1 samples. Compared to other features, this feature is more affected by packers (30%). This makes sense as packers often hide the import table of the original code, replacing it with a potentially smaller table. This is reflected in the 10 largest clusters, which are all MCs. Five of them are likely due to known EXE-building tools: InnoSetup (2), aut2exe (1), and the UPX (1) and Themida (1) packers. The import table for the UPX cluster has 21 imports and the one from Themida only two. Another four clusters have very small import tables with 1–4 imports and contain detections for multiple packers. Thus, different packers may generate the same small import table. The final cluster is one of the previously mentioned with GT errors.

We conclude that *imphash* is not very good at grouping samples into families. Not only it is affected by EXE-building tools, but it also groups samples built using different packers that produce the same small import tables. And since packers are very common among malware authors, this is a very severe limitation. Removing samples with very small import tables can ameliorate the impact of packers, but it would remove a large number of samples and would not affect MCs caused by other tools, e.g., the mentioned aut2exe MC has 522 imports, and the InnoSetup MCs 99–137.

Richpe. This feature has the third lowest precision (0.887). Only 25% of its 656 MCs have a possible explanation with packers (14%) being the most common explanation. Out of the 10 largest clusters, all are MCs and all have a small number of entries (1–13) in the RichPE header. One MC is likely caused by the NSIS installer. For the rest, there is no likely cause and we observe a mixture of packers and other tools. This likely indicates collisions where different EXE-building tools happen to have used the same compilers to build their modules. It is interesting to note that in this case collisions are also observed with samples where no EXE-building tool is detected.

Similar to *imphash*, we conclude that the *richpe* hash is not very good at grouping malware samples into families, suffering from collisions between different EXE-building tools and other malicious samples.

Icons. *icon_dhash* has lower precision (0.856) and higher recall (0.203) than *icon_hash* (0.888 and 0.186, respectively), as it groups samples that have visually similar, but not identical, icons, incorrectly grouping unrelated samples. A small ratio of MCs has a likely explanation: 16% for *icon_hash* and 14% for *icon_dhash*. Of the 10 largest clusters for *icon_hash*, all are MCs, although one seems to identify *zbot* (352 samples) with one incorrectly labeled *virut* sample. The other nine are due to the icons in Figure 1. These are common icons, not specific to a single family. Thus, their MCs contain samples from many (25–100) families identified as using different EXE-building tools and no tools at all.

We conclude that *icon_hash* works better than *icon_dhash* for grouping samples into families, but both suffer from

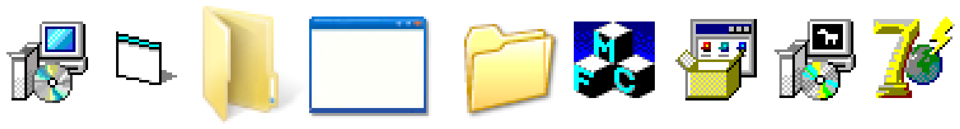
Fig. 1. Icons responsible for the 9 largest MCs using *icon_hash*.

TABLE VII
CLUSTERING PRECISION ON THE SUPERSET DATASET BEFORE (ORIGINAL) AND AFTER (NOSINGLETONS) REMOVING THE SAMPLES BELONGING TO SINGLETON CLUSTERS FOR EACH FEATURE.

Feature	Original	NoSingletons	Delta
authentihash	0.997	0.939	0.059
cert_subject	0.991	0.911	0.080
cert_thumbprint	0.994	0.934	0.060
icon_dhash	0.856	0.708	0.149
icon_hash	0.888	0.734	0.154
imphash	0.873	0.838	0.035
pehash	0.963	0.946	0.017
richpe	0.887	0.784	0.103
ssdeep	0.967	0.941	0.026
tlsh	0.959	0.930	0.029
vhash	0.939	0.918	0.020
avclass	0.909	0.906	0.003

common icons that are not specific to a family and are responsible for 9 out of 10 of the largest clusters.

AVClass. AVClass has the lowest ratio (4%) of MCs with a possible explanation. This makes sense as AVClass produces much larger clusters (81.5 samples on average) due to its ability to link samples of the same family considered dissimilar by individual features. Thus, the analysis features should rarely be able to explain an AVClass MC. Most likely, the possible explanations for those 4% MCs capture correlation rather than causality.

No-overlay hash. Interestingly, in MalPE 156 no-overlay hashes are common between samples of different families. The extreme case is a no-overlay hash that matches no-overlay samples in 12 families. This no-overlay hash is known to VirusTotal and has 1/69 detections at the time of writing. The corresponding file is *7zS.sfx*, a template file used to build 7-zip self-extracting archives as Windows executables that can self-extract their contents upon execution. It is possible to create a self-extracting archive with 7zip by concatenating *7zS.sfx* with a text configuration file and a compressed archive [5]. In such a self-extracting archive, the PE file is *7zS.sfx*, and the configuration file and the archive are both in an overlay. Another no-overlay hash matches 6 families. It corresponds to *Default.sfx*, a file used by WinRAR to generate self-extracting files. These cases show that the use of self-extracting archives is common in malware families that can use them to create installers when they need to distribute multiple files, as well as for obfuscation since analyzing these executables requires analyzing the compressed data in the overlay.

Impact of singleton clusters. To better understand how clustering performance is influenced by singleton clusters, we recompute precision after excluding them. This isolates the contribution of clusters that group multiple samples, where errors are more likely to occur and where grouping ability is meaningfully tested.

Features like *ssdeep*, *pehash*, and *tlsh* maintain high precision even without singletons (0.941, 0.946, and 0.930 respectively), with minimal drops ($\leq$ 0.03). *vhash* and *imphash* show similarly stable behavior, with adjusted precision above 0.90 and low delta values. This indicates that their clustering performance is not artificially boosted by singleton clusters. In contrast, certificate-based features such as *cert_subject* and *cert_thumbprint* show notable drops in precision (8.0 and 6.0 points, respectively), indicating that a significant portion of their reported scores stems from singleton clusters. Visual features such as *icon_dhash* and *icon_hash* show the most substantial losses (14.9 and 15.4 points) and also yield the lowest adjusted precision (0.708 and 0.734), confirming their limited utility for consistent family-level clustering. Interestingly, although *authentihash* sees a notable drop (from 0.997 to 0.939), this does not necessarily indicate poor clustering behavior. As previously discussed, this feature often isolates samples that share the same true origin but were assigned inconsistent family labels. It may therefore reflect labeling errors rather than over-segmentation, making it valuable for identifying inconsistencies in ground truth annotations. Finally, *avclass*, which assigns family names without relying on similarity-based grouping, is naturally unaffected by singleton exclusion, showing only a negligible change.

VII. DISCUSSION

This section discusses the impact of our results on malware family clustering and potential avenues for improvements.

Feature effectiveness. The similarity features analyzed can broadly be divided into three groups. First are the *authentihash* and the leaf certificate features, which have nearly perfect precision (over 99%) but limited grouping ability. We suggest that these features be used during dataset construction to spot potential errors in the GT labels. Next come structural hashes (*pehash*, *vhash*) and whole-file fuzzy hashes (*ssdeep*, *ssdeep*) with precisions 94%–97%. These features are impacted by EXE-building tools, which may cause them to capture similarity not due to family characteristics but rather to the EXE-building technology itself, thus causing samples from unrelated malware families to be placed together in MCs. For instance, for SSDeep and TLSH up to 45% of MCs were likely caused by EXE-building tools. In this group, *pehash* seems the best choice for grouping samples into families as it has the highest precision and average cluster size. Finally, the remaining 4 features (*imphash*, *richpe*, both

TABLE VIII
CLUSTERING PRECISION ON THE SUPERSET DATASET BEFORE (ORIGINAL) AND AFTER (NOBUILT) PLACING THE 35,766 SAMPLES IDENTIFIED AS BEING BUILT WITH EXE-GENERATING TOOLS IN SINGLETON CLUSTERS.

Feature	Original	NoBuilt	Delta
authentihash	0.997	1.000	0.003
cert_subject	0.991	0.998	0.007
cert_thumbprint	0.994	0.999	0.005
icon_dhash	0.856	0.949	0.093
icon_hash	0.888	0.963	0.075
imphash	0.873	0.976	0.103
pehash	0.963	0.988	0.025
richpe	0.887	0.959	0.072
ssdeep	0.967	0.991	0.024
tlsh	0.959	0.987	0.028
vhash	0.939	0.982	0.043
avclass	0.909	0.942	0.033

icon features) have precision below 90%. They are affected by EXE-building tools, but also show collisions in values due to other factors (e.g., short tables or generic icons). These features can introduce significant errors in family clustering.

Impact of EXE-Building Tools. A recurring theme in our analysis was the impact of EXE-building tools. These tools may cause features to capture similarity not due to family characteristics but rather to the EXE-building technology itself, thus causing samples from unrelated malware families to be clustered together. For instance, for SSDeep and TLSH up to 45% of MCs were likely caused by EXE-building tools. Packers were the most likely cause for MCs across different features, potentially explaining 20% of all MCs, followed by SFX archives (5.6%), Installer (4%), and script-building tools (4%).

Feature pre-processing. A potential fix to address the limits of the similarity features would be to avoid using them in cases known to be problematic. Table VIII measures the improvement in feature precision if we place in singleton clusters samples identified as being produced by EXE-building tools. This is a generic fix that can be applied to all similarity features. The top three features by precision (*authentihash* and the certificate features) show very limited improvement (<1%) since they are originally little affected by EXE-building tools. In contrast, *imphash* shows a precision improvement of 10.3% The icon features and *richpe* still maintain lower precision (94.9%–96.3%) indicating that collisions unrelated to EXE-building tools still happen often.

Another option is applying specific pre-processing to selected features. For example, we showed that if we avoid extracting leaf certificate features from samples with invalid certificate chains, the precision of the certificate features increased by 2% for *cert_thumbprint* and 4% for *cert_subject*. Similarly, if we place in singletons samples with an import table with less than 10 entries, the *imphash* precision improves by 3.4% from 0.873 to 0.907. However, this is lower than the precision obtained by especially handling samples produced with EXE-building tools (0.976) showing that collisions also happen on larger import tables. Another potential fix is ignoring generic icons in both icon features. However, identifying generic icons is a challenge in itself.

Ground truth errors. We identified several cases where MCs are likely due to GT errors, rather than to limits of the similarity features. Most cases are aliases identifying the same family, which can happen within a dataset and across datasets. For example, in the *authentihash* analysis we observed two groups of aliases within MalPE and we also spotted likely aliases in MOTIF, e.g., *ligsetrac* and *skimer*. But, we also observe aliases across datasets such as MOTIF using *kronos* where MalPE uses *kronosbot*. We have been reporting such cases to the AVClass developers and hope that the identified aliases will be incorporated into that tool. We also spot cases where individual samples seem incorrectly labeled. However, verifying these errors is harder and may require significant manual analysis.

Feature-Specific observations. In summary, each feature exhibits distinct strengths and weaknesses in its ability to group malware samples effectively. Authentihash, with its near-perfect precision, stands out as a valuable tool for identifying errors in ground truth labeling during dataset construction despite its limited grouping capabilities. SSDeep and TLSH demonstrate high clustering performance but primarily capture similarities introduced by EXE-building tools, limiting their efficacy in identifying actual malicious similarities. Pehash outperforms fuzzy hashes in grouping malware families but remains susceptible to EXE-building tool artifacts. Imphash and richpe hashes perform poorly due to vulnerabilities to EXE-building tools and packer-induced noise, with imphash additionally challenged by grouping across diverse packer outputs. Vhash fares worse than pehash, exhibiting lower precision and higher collision rates. Certificate-based features show high precision and robustness against EXE-building tools, though their utility is constrained by the misuse of benign certificates, requiring careful extraction protocols. Lastly, icon-based features, while useful, suffer from the influence of benign icons that obscure family-specific patterns.

VIII. CONCLUSION

This study presents a systematic evaluation of eleven widely used static features for malware clustering, providing not only a comparative performance analysis but also an investigation into the structural causes of clustering failures. By analyzing three large-scale datasets and identifying how packaging artifacts and EXE-building tools impact clustering quality, we offer insights that go beyond raw performance metrics and inform the refinement of clustering pipelines.

Our findings highlight that certain features, while yielding high clustering precision, may be heavily influenced by superficial attributes such as packing tools or icon reuse. This underscores the importance of interpreting high-precision clusters with caution, as they may reflect operational similarity rather than true family relationships. Our study encourages future research to complement feature evaluation with failure analysis to better understand these edge cases.

These results have significant implications for both academic research and operational malware triage. In particular, the choice of static features should be guided not only by empirical accuracy but also by an understanding of what underlying properties those features capture and how those align with the intended clustering objective.

Future work should focus on better characterizing the presence and influence of EXE-building tools in malware datasets, in order to assess the generalizability of clustering results across corpora. Additionally, the refinement of preprocessing techniques—such as filtering short import tables or removing highly reused icons—could further improve feature robustness and help reduce noise in downstream clustering.

Overall, our work emphasizes the need for critical examination of static features and encourages the malware analysis community to adopt more diagnostic approaches in the evaluation of clustering quality.

A. Limitations

While this study provides a comprehensive analysis of widely used static features for malware clustering, several limitations must be acknowledged. The evaluation relies on ground truth family labels derived from tools like AVClass or from datasets constructed via automated methods. As the exact heuristics used by antivirus engines are proprietary and not disclosed, there may be discrepancies between the criteria these labels reflect and the similarity captured by the features under study. This misalignment can influence the apparent precision of clustering outcomes and complicate the interpretation of results.

Our analysis primarily focuses on precision-oriented metrics, as the goal of this work is to identify and understand structural similarities across different malware families. Precision emphasizes the internal consistency of clusters and is particularly suited to evaluating whether a given feature produces meaningful groupings that reflect shared traits. While recall and other measures capture complementary aspects, such as coverage or fragmentation, they are less aligned with our objective of analyzing why unrelated samples may appear clustered. Nevertheless, broader evaluation frameworks could be considered in future work, especially for use cases like malware triage or campaign tracking.

Finally, while we analyze the most commonly used static features, other types of representations remain unexplored. These include image-based encodings of binaries, as used in CNN-based malware classification [41], [28], [17], [65], as well as dynamic or hybrid features that capture behavioral aspects. Such alternative features may yield different clustering behaviors and could complement the perspective offered by static analysis alone. Moreover, while this work analyzes features in isolation, combining multiple static features, especially those that capture complementary signals, could further improve clustering quality. However, such combinations would effectively define new clustering algorithms or similarity measures, and are thus considered out of the scope of this work.

Acknowledgments

This work has benefited from two government grants managed by the National Research Agency under France 2030 with the references “ANR-22-PECY-0007” (DefMal) and “ANR-23-IAS4-0001” (CKRISP). This work has also been partially supported by MICIU/AEI/10.13039/501100011033 through grants PID2022-142290OB-I00 (ESPADA), TED2021-132464B-I00 (PRODIGY), CEX2024-001471, and RED2024-154240-T (EMACS). These projects are co-funded by the European Union EIE and NextGeneration EU/PRTR Funds.

References

[1] Tracking malware with import hashing, https://www.mandiant.com/resources/tracking-malware-import-hashing
[2] Pe identifier (peid) (2024), https://github.com/wolfram77web/app-peid
[3] Hash and family of each sample. https://raw.githubusercontent.com/eurecom-s3/DecodingMLSecretsOfWindowsMalwareClassification/main/dataset/malware (Accessed November 17, 2025)
[4] MOTIF Dataset. https://github.com/boozallen/MOTIF (Accessed November 17, 2025)
[5] How to make a self extracting archive that runs your setup.exe with 7zip -sfx switch. https://ntsblog.homedev.com.au/index.php/2015/05/14/self-extracting-archive-runs-setup-exe-7zip-sfx-switch/ (May 2015)
[6] Ali, M., Hagen, J., Oliver, J.: Scalable malware clustering using multi-stage tree parallelization. In: 2020 IEEE International Conference on Intelligence and Security Informatics (ISI). pp. 1–6. IEEE (2020)
[7] Anderson, B., Roth, D., McGrew, D.: Improving malware classification: Bridging the static/dynamic gap. In: Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence (AISec) (2012)
[8] Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J.: Automated Classification and Analysis of Internet Malware. In: International Symposium on Recent Advances in Intrusion Detection (2007)
[9] Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E.: Scalable, Behavior-Based Malware Clustering. In: Network and Distributed System Security Symposium (2009)
[10] Botacin, M., Moia, V.H.G., Ceschin, F., Henriques, M.A.A., Grégio, A.: Understanding uses and misuses of similarity hashing functions for malware detection and family clustering in actual scenarios. Forensic Science International: Digital Investigation **38**, 301220 (2021)
[11] Chikapa, M., Namanya, A.P.: Towards a fast off-line static malware analysis framework. In: 2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW). pp. 182–187. IEEE (2018)
[12] Cozzi, E., Graziano, M., Fratantonio, Y., Balzarotti, D.: Understanding Linux Malware. In: IEEE Symposium on Security and Privacy (2018)
[13] Dambra, S., Han, Y., Aonzo, S., Kotzias, P., Vitale, A., Caballero, J., Balzarotti, D., Bilge, L.: Decoding the Secrets of Machine Learning in Malware Classification: A Deep Dive into Datasets, Feature Extraction, and Model Performance. In: ACM Conference on Computer and Communications Security (2023)
[14] French, D., Casey, W.: 2 fuzzy hashing techniques in applied malware analysis. Results of SEI Line-Funded Exploratory New Starts Projects p. 2 (2012)
[15] Gheorghescu, M.: An automated virus classification system. In: Virus bulletin conference (2005)
[16] Gibert, D., Mateu, C., Planes, J., Vicens, R.: Using convolutional neural networks for classification of malware represented as images. Journal of Computer Virology and Hacking Techniques **15**, 15–28 (2019)
[17] Han, K., He, Y., Zhang, Z., Liang, Z., Tang, Y., Zhang, Z., Li, Z., Liu, P.: MalDA: Robust malware detection using attention-based graph neural networks. In: Proceedings of the 27th ACM Conference on Computer and Communications Security (CCS). pp. 387–400. ACM (2020). https://doi.org/10.1145/3372297.3417890
[18] Haq, I.U., Chica, S., Caballero, J., Jha, S.: Malware lineage in the wild. Computers & Security **78**, 347–363 (2018)
[19] horsicq: Detect It Easy. https://github.com/horsicq/Detect-It-Easy, [Online; November 17, 2025]

[20] Hu, X., Shin, K.G., Bhatkar, S., Griffin, K.: MutantX-S: Scalable Malware Clustering Based on Static Features. In: USENIX Annual Technical Conference (2013)

[21] Ippolito, L.: A Framework for the Analysis of File Infection Malware. Master's thesis, Politecnico Di Torino, Torino, Italy (March 2024)

[22] Jang, J., Brumley, D., Venkataraman, S.: Bitshred: feature hashing malware for scalable triage and semantic analysis. In: ACM Conference on Computer and Communications Security (2011)

[23] Joyce, R.J., Amlani, D., Nicholas, C., Raff, E.: MOTIF: A large malware reference dataset with ground truth family labels. In: Workshop on Artificial Intelligence for Cyber Security (2022), https://github.com/boozallen/MOTIF

[24] Joyce, R.J., Bilzer, K., Burke, S.: Malware attribution using the rich header (2019)

[25] Kaczmarczyck, F., Grill, B., Invernizzi, L., Pullman, J., Procopiuc, C.M., Tao, D., Benko, B., Bursztein, E.: Spotlight: Malware Lead Generation at Scale. In: Annual Computer Security Applications Conference (2020)

[26] Kim, D., Kwon, B.J., Dumitraş, T.: Certified malware: Measuring breaches of trust in the windows code-signing pki. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1435–1448 (2017)

[27] Kim, J.S., Jung, W., Kim, S., Lee, S., Kim, E.T.: Evaluation of image similarity algorithms for malware fake-icon detection. In: 2020 International Conference on Information and Communication Technology Convergence (ICTC). pp. 1638–1640. IEEE (2020)

[28] Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C.: Deep learning for classification of malware system call sequences. In: 9th International Conference on Malicious and Unwanted Software (MALWARE). pp. 61–68. IEEE (2016). https://doi.org/10.1109/MALWARE.2016.7808511

[29] Kornblum, J.: Identifying Almost Identical Files Using Context Triggered Piecewise Hashing. Digital Investigation **3**, 91–97 (Sep 2006)

[30] Kotzias, P., Matic, S., Rivera, R., Caballero, J.: Certified pup: abuse in authenticode code signing. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 465–478 (2015)

[31] Kruegel, C., Robertson, W., Valeur, F., Vigna, G.: Static Disassembly of Obfuscated Binaries. In: USENIX Security Symposium (2004)

[32] Li, P., Liu, L., Gao, D., Reiter, M.K.: On challenges in evaluating malware clustering. In: International Symposium on Recent Advances in Intrusion Detection (2010)

[33] Li, S., Ming, J., Qiu, P., Chen, Q., Liu, L., Bao, H., Wang, Q., Jia, C.: Packgenome: Automatically generating robust yara rules for accurate malware packer detection. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 3078–3092 (2023)

[34] Li, Y., Sundaramurthy, S.C., Bardas, A.G., Ou, X., Caragea, D., Hu, X., Jang, J.: Experimental study of fuzzy hashing in malware clustering analysis. In: 8th workshop on cyber security experimentation and test (cset 15) (2015)

[35] Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A., Balzarotti, D.: Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In: Network and Distributed System Security (NDSS) Symposium, NDSS. vol. 20 (2020)

[36] Mohaisen, A., Alrawi, O., Mohaisen, M.: Amal: High-fidelity, behavior-based automated malware analysis and classification. In: Proceedings of the 7th International Conference on Information Systems Security (ICISS) (2015)

[37] Naik, N., Jenkins, P., Savage, N., Yang, L., Boongoen, T., Iam-On, N., Naik, K., Song, J.: Embedded yara rules: strengthening yara rules utilising fuzzy hashing and fuzzy rules for malware analysis. Complex & Intelligent Systems **7**, 687–702 (2021)

[38] Naik, N., Jenkins, P., Savage, N., Yang, L., Naik, K., Song, J., Boongoen, T., Iam-On, N.: Fuzzy hashing aided enhanced yara rules for malware triaging. In: IEEE Symposium Series on Computational Intelligence (2020)

[39] Namanya, A.P., Awan, I.U., Disso, J.P., Younas, M.: Similarity hash based scoring of portable executable files for efficient malware detection in iot. Future Generation Computer Systems **110**, 824–832 (2020)

[40] Nappa, A., Rafique, M.Z., Caballero, J.: The MALICIA Dataset: Identification and Analysis of Drive-by Download Operations. International Journal of Information Security **14**(1), 15–33 (February 2015)

[41] Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images: Visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec). pp. 1–7. ACM (2011). https://doi.org/10.1109/VISSOF.2011.6069446

[42] Oliver, J., Ali, M., Hagen, J.: Hac-t and fast search for similarity in security. In: International Conference on Omni-layer Intelligent Systems (2020)

[43] Oliver, J., Cheng, C., Chen, Y.: Tlsh–a locality sensitive hash. In: Cybercrime and Trustworthy Computing Workshop (2013)

[44] Pagani, F., Dell'Amico, M., Balzarotti, D.: Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis. In: ACM Conference on Data and Application Security and Privacy (2018)

[45] Perdisci, R., Lee, W., Feamster, N.: Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In: USENIX Symposium on Networked Systems Design and Implementation (2010)

[46] Perdisci, R., ManChon, U.: VAMO: Towards a Fully Automated Malware Clustering Validity Analysis. In: Annual Computer Security Applications Conference (2012)

[47] Poslušnỳ, M., Kálnai, P.: Rich headers: Leveraging this mysterious artifact of the pe format. Virus Bulletin (October 2019)

[48] Rafique, M.Z., Caballero, J.: FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In: Symposium on Research in Attacks, Intrusions and Defenses (2013)

[49] Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic Analysis of Malware Behavior using Machine Learning. Journal of Computer Security **19**(4) (2011)

[50] Sebastián, M., Rivera, R., Kotzias, P., Caballero, J.: AVClass: A Tool for Massive Malware Labeling. In: International Symposium on Research in Attacks, Intrusions, and Defenses (2016)

[51] Sebastián, S., Caballero, J.: AVClass2: Massive Malware Tag Extraction from AV Labels. In: Annual Computer Security Applications Conference (2020)

[52] ssdeep - Fuzzy hashing program, https://ssdeep-project.github.io/ssdeep/index.html

[53] Ugarte-Pedrero, X., Balzarotti, D., Santos, I., Bringas, P.G.: SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers. In: IEEE Symposium on Security and Privacy (2015)

[54] Ugarte-Pedrero, X., Graziano, M., Balzarotti, D.: A close look at a daily dataset of malware samples. ACM Transactions on Privacy and Security (TOPS) **22**(1), 1–30 (2019)

[55] Ul Haq, I., Caballero, J.: A Survey of Binary Code Similarity. ACM Computing Surveys **54**(3) (April 2021)

[56] Venkatraman, S., Alazab, M., Vinayakumar, R.: A hybrid deep learning image-based analysis for effective malware detection. Journal of Information Security and Applications **47**, 377–389 (2019)

[57] VirusTotal. https://www.virustotal.com/ (Accessed November 17, 2025)

[58] (2021), https://developers.virustotal.com/reference/files

[59] Wageh, A.: Automating The Analysis Of An AutoIT Script That Wraps A Remcos RAT. https://amgedwageh.medium.com/analysis-of-an-autoit-script-that-wraps-a-remcos-rat-6b5b66075b87 (January 2022)

[60] Webster, G.D., Kolosnjaji, B., von Pentz, C., Kirsch, J., Hanif, Z.D., Zarras, A., Eckert, C.: Finding the needle: A study of the pe32 rich header and respective malware triage. In: Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14. pp. 119–138. Springer (2017)

[61] Wicherski, G.: pehash: A novel approach to fast malware clustering. In: 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET) (2009)

[62] Xiao, G., Li, J., Chen, Y., Li, K.: Malfcs: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. Journal of Parallel and Distributed Computing **141**, 49–58 (2020)

[63] Yan, J., Qi, Y., Rao, Q.: Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks **2018**(1), 7247095 (2018)

[64] Yang, W., Gao, M., Chen, L., Liu, Z., Ying, L.: Recmal: Rectify the malware family label via hybrid analysis. Computers & Security **128**, 103177 (2023)

[65] Zhu, B., Wang, Y., Du, X.: Image-based malware classification using 2d convolutional neural networks and transfer learning. Journal of Computer Virology and Hacking Techniques **16**(2), 173–182 (2020). https://doi.org/10.1007/s11416-019-00331-2