Understanding Uses and Misuses of Similarity Hashing Functions for Malware Detection and Family Clustering in Actual Scenarios

Marcus Botacin[1] Vitor Hugo Galhardo Moia[2,3] Fabricio Ceschin[1]
Marco A. Amaral Henriques[3] André Grégio[1]

[1]Federal University of Paraná (UFPR)
{mfbotacin,fjoceschin,gregio}@inf.ufpr.br
[2] Samsung R&D Institute Brazil
vitor.m@samsung.com
[3] University of Campinas (UNICAMP)
maah@unicamp.br

Abstract

An everyday growing number of malware variants target end-users and organizations. To reduce the amount of individual malware handling, security analysts apply techniques for finding similarities to cluster samples. A popular clustering method relies on similarity hashing functions, which create short representations of files and compare them to produce a score related to the similarity level between them. Despite the popularity of those functions, the limits of their application to malware samples have not been extensively studied so-far. To help in bridging this gap, we performed a set of experiments to characterize the application of these functions on long-term, realistic malware analysis scenarios. To do so, we introduce SHAVE, an ideal model of similarity hashing-based antivirus engine. The evaluation of SHAVE consisted of applying two distinct hash functions (ssdeep and sdhash) to a dataset of 21 thousand actual malware samples collected over four years. We characterized this dataset based on the performed clustering, and discovered that: (i) smaller groups are prevalent than large ones; (ii) the threshold value chosen may significantly change the conclusions about the prevalence of similar samples in a given dataset; (iii) establishing a ground-truth for similarity hashing functions comparison has its issues, since the clusters originated from traditional AV labeling routines may result from a completely distinct approach; (iv) the application of similarity hashing functions improves traditional AVs' detection rates by up to 40%; and finally, (v) taking specific binary regions into account (e.g., instructions) leads to better classification results than hashing the entire binary file.

1 Introduction

Antiviruses (AVs) have been the main line of defense against malware infections in the last years, but even the AV industry struggles to provide countermeasures for the exponentially growing number of malware variants released daily [22]. AV companies usually handle the huge amount of newly discovered malware by grouping samples into clusters, then applying the same detection processes to each of them. Thus, companies save time and prevent their analysts from analyzing plenty of similar individual samples. This procedure of complexity reduction is backed by the fact that many samples are created by the same attackers with the same "development model", which incur in resulting malware that exhibit similar constructs [4].

Identifying malware similarity is as challenging as it is important for defense purposes. Many different approaches have been proposed for similarity identification over time. A popular similarity identification technique is similarity hashing (a.k.a., Approximate Matching [18]), which produces small and compact object representations called *digests*. The comparison of two objects' digests using these types of functions produces a value (score) that indicates the level of similarity between those objects; a given threshold determines whether the objects are considered similar to each other or not. Similarity hashing functions became popular due to their greater speed in comparison to the aforementioned related work, thus being implemented by many security companies in their products, servers, and endpoints (e.g., TrendMicro relies on similarity hashing for IoT malware clustering [43], whereas Google applies it for finding malicious apps in its Play Store [28]).

Despite their popularity, the academic literature about similarity hashing functions is still limited, either in number or in the broadness of their use in malware detection. For instance, Pagani et al. [56] presented an extensive evaluation on the application of hash functions for many binary analysis tasks, but the specific case of malware variants was little explored. No available work focuses on the drawbacks of malware analysis, such as threshold definition issues. The literature also does not discuss the limitations, development opportunities, and challenges of similarity hashing application to actual malware samples in real scenarios. Therefore, we decided to investigate these possibilities and fill these gaps with a methodological approach for the application of similarity hashing functions for malware classification and clustering.

We investigate the application of similarity hashing functions for three distinct, independent tasks: (i) characterization of real datasets; (ii) malware detection; and (iii) malware clustering in families. For each one of them, we highlight similarity hashing function's strong and weak points. For all tasks, since our goal is to evaluate the functions in real scenarios, we used a dataset of 21 thousand distinct real-world malware samples collected during seven years from real infected machines.

For the characterization task, we leverage distinct setups of hashing functions (`ssdeep` and `sdhash`) and thresholds (from 0 up to 100%), such that we can explore the best parameters for performing the task at hand. We report a set of exploratory results on the application of hashing functions to malware samples, such as the number of clustered samples, their sizes, and the impact of distinct threshold values on clusters definition. We successfully identified multiple clusters in the real-world dataset, some of which containing up to a hundred samples. We discovered that distinct threshold values might completely change the dataset characterization outcomes by indicating that it presents a low or a high ratio of similar samples. Even worse, there is currently no guideline for such threshold definition, such that distinct researchers choose values in an ad-hoc manner.

For the detection task, we introduce SHAVE (Similarity Hashing AntiVirus Engine), an ideal model of a cloud-based AV with similarity hashing capabilities to evaluate its application to malware samples. By leveraging SHAVE, we discovered that traditional AVs' detection rates can be increased by up to 40% if the similarity between detected and undetected samples is taken into account. We also discovered that AVs cloud-based architectures might allow that more complex similarity identification routines to be put into practice by applying them only to the cluster's centroids, instead of to all samples.

For the malware classification task, we compared similarity hashing function's and traditional AntiVirus' (AVs) clustering capabilities, since AVs are the most popular solutions for labeling malware. We identified a difficulty to establish a ground-truth for the number and size of clusters reported by the similarity hashing functions. We highlight that it is difficult to compare SHAVE and traditional AVs — the closest-related malware detection solution to this work — since the latter generate clusters based on completely distinct metrics (e.g., behavioral heuristics). We conclude that malware classification is the hardest task to be performed either by using AVs or similarity hashing functions, which is often reflected in the lower metrics scores (accuracy, precision, so on) of these methods, as reported in the paper. In an attempt to increase the classification capabilities, we investigate distinct strategies, such as considering only specific binary excerpts and/or sections. For instance, we investigate the impact of common blocks [47], i.e., common structures that repeat in many objects of the same and different file types, to the similarity scores. We discovered that considering only the 70% most prevalent binary's instructions slightly increases the overall metrics scores (e.g., accuracy, precision, recall).

Therefore, our results suggest that (i) the adoption of similarity hashing functions is a promising candidate technique to support the development of the next generation of AVs intended to operate along with large-scale demands; but (ii) their application should care about the drawbacks we describe in this study.

Our major goal with this case study is to discuss and suggest mechanisms to increase the confidence and reliability of the malware classification procedures using similarity hashing functions used by malware analysts and forensic experts. In this sense, our contributions in this paper are:

- We propose SHAVE, which presents the concept of a similarity hashing-based AV engine operating solely from a cloud server and with a centralized malware definition repository.
- We evaluate the limits and benefits of applying similarity hashing functions for malware detection and classification in real-world scenarios, investigate the impact of removing common blocks from the similarity scoring, and show the issues of using AV labels as ground-truth from malware clustering.
- We show that using AV labels as ground-truth for malware families clustering is challenging, since labels come from other techniques than similarity hashing functions, producing distinct results.
- We investigate the impact of adapting a forensic common blocks removal approach to the malware similarity identification context so as to allow the most discriminant binary instructions to influence more on the final similarity score rather than common instructions.
- We pinpoint and discuss challenges and opportunities for future work on malware detection and similarity clustering.

The remainder of this paper is organized as follows: In Section 2, we present background information on how similarity hashing functions work; In Section 3, we present our experimental methodology and the concept of a similarity hashing-based AntiVirus; In Section 4, we evaluate the application of similarity hashing functions for malware detection and classification in real-world scenarios; In Section 5, we revisit our findings to propose guidelines for the application of similarity hashing function; In Section 6, we discuss the implications of our findings; In Section 7, we present related work to better position our contributions; Finally, we draw our conclusions in Section 8.

2 Background

This work's goal is to evaluate the application of similarity hashing to speed up malware clustering procedures. Therefore, we here present background information on similarity hashing that supported the development of this work and describe the clustering tasks we are tackling using them.

2.1 Similarity Hashing

Traditional hash functions (e.g., MD5, SHA-1, SHA-2) aim to create small and compact object representations (*a.k.a.*, digests) for their unique identification. The main characteristic of hashes is that flipping a single bit in the input drastically changes the output, making possible only the detection of identical objects. To tackle traditional hash limitation, similarity hashing functions were developed to create short object representations and allow the identification of similarity between objects. A small change in the input will reflect in a minor and located variation in the digest, allowing similar objects to be correlated. Another difference between hashes and similarity hashing is that, while hashes produce binary answers for the comparison of two objects (they are identical or not), the other one uses special comparison functions that provide a confidence measure (score) about the similarity of two objects. This score can be in a fixed interval (from 0 to 100) or express dissimilarity (from 0 up to a given limit). A score of 100 (or another value representing the perfect value - maximum similarity) produced by similarity hashes does not mean that the two compared objects are identical in every single byte, but that they have a high degree of similarity.

The digest generation process [45] is composed by a feature extraction (in this case, feature is defined as a sequence of bytes extracted from the object) and a filtering step to reduce the number of produced features. This happens because some tools sometimes extract overlapping features and/or some features are weak and should not be considered due to an increase in false positives (e.g., a sequence of zeros). The interpretation of the final result varies according to the tool; some use fixed threshold values to declare similarity; others produce percentage values.

Similarity hashing is widely applied in the digital forensic field, where practitioners use them to find similar data in multiple contexts [61, 31], including the location of malware variants, the topic of this work. Although these functions are formally known as Approximate Matching [18], we opted for using the term similarity hashing for the rest of this work since it is most known in the malware community.

The Multiple Similarity Hashing Functions. Many tools implement the concepts of similarity hashes and it is hard to cover all of them in a single research effort. Therefore, we limited our analysis to the most popular solutions. Table 1 shows a sampling of the most recent papers available in popular publisher's (ACM, IEEE, Elsevier) repositories. Our findings suggest `ssdeep` and `sdhash` as the targets of most research works in the literature. Besides, these two functions were also scrutinized in many research papers (see next paragraphs) and are well accepted by the community.

Table 1: **Recently Published Works.** ssdeep and sdhash are the most popular functions.

Work	**ssdeep**	**sdhash**	**TLSH**	**myhash**	**Lempel-ziv**
Shiel et al. [66]	✓				
Sarantinos et al. [64]	✓	✓			
Pagani et al. [56]	✓	✓			
Azab et al. [4]	✓	✓	✓		
Naik et al. [51]	✓	✓		✓	
Raff et al. [59]	✓	✓			✓

In the following paragraphs, we summarize the use and the operation of these two well known tools (ssdeep and sdhash) which are the target of constant research and the focus of this paper. A detailed analysis of their internal aspects is presented in [42].

ssdeep is a popular similarity hashing function used in multiple contexts. It implements the Content Triggered Piecewise Hashing (CTPH) concept to detect content similarity at the bytewise level. `ssdeep` creates variable size blocks with the aid of a rolling hash algorithm to determine when blocks start and stop (set boundaries) based on a (random) value obtained from a fixed-size window that moves through the input byte-by-byte. Next, all blocks produced are hashed using an FNV hash function [52], and the six least significant bits of each hash are encoded in a Base64 character; the digest is the concatenation of all characters. The similarity is assessed with `ssdeep` by using an Edit Distance function [68] that counts the minimum number of operations required to transform one digest into the other, producing a value (score) indicating similarity; it ranges from 0 (dissimilarity) to 100 (perfect match). No threshold is used by ssdeep; all comparisons with score > 0 are considered as similar. More information about the whole process of creating and comparing digests can be found in [36].

sdhash is another similarity hashing function that operates in the bytewise level to detect similarity between objects [60]. In short, `sdhash` aims to identify statistically improbable features in objects (i.e., unique ones) and mapping them into similarity digests. A feature in this context is a β-bytes sequence (64 by default) extracted from the object, and those which have the lowest empirical probability of being encountered by chance (based on their Shannon entropy) are selected to represent the object. Next, all selected features are hashed using SHA-1 and stored into bloom filters [10]. Each filter can store at most 160 features; in case a filter reaches its capacity, a new one is created. The final object digest is the sequence of all bloom filters. The similarity of two digests is measured by comparing the set of bloom filters against each other; a normalized score is returned in the range of 0 (no similarity) to 100 (very similar or identical). Roussev, V. and Quates, C. [63] suggests the use of a threshold t value of 21 to define when objects are similar since comparisons with lower scores tend to produce many false positives, although this threshold was not evaluated in the malware detection context. Many studies have been developed to improve `sdhash` efficiency [34] and precision [47, 45]. In this work, we adopt a new version of `sdhash` called `J-sdhash` [45], since it adopts a new comparison function that is more precise and whose results are easier to interpret, as they are expressed in the terms of the real similarity shared between objects.

2.2 Malware Clustering & Similarity

The presented functions can be applied to many classification tasks and over any type of binary. Malware analysis is one of the most popular tasks for similarity hashing functions and therefore our focus in this work. Malware classification tasks can be divided into three types:

1. **Malware Family Clustering:** In this task, a set of binaries reported as malicious (with no malware family label) are classified into clusters of similar binaries (malware families). The goal is to label them into N clusters that maximize their similarities, generating family labels for future classification and remediation tasks. Note that this step is similar to unsupervised learning in machine learning, given that there are no labels involved in the process of generating labels for unknown samples [8].

2. **Malware Detection:** In this task, unknown binaries are classified into a cluster of either malware or goodware samples. The goal is to identify whether the unknown binary is malicious or not based on a set of known malware and goodware samples, similar to a supervised learning process [8].

3. **Family Classification:** In this task, a binary reported as malicious is classified into clusters of similar known labeled malware binaries (malware families). This task's goal is to separate malware samples belonging to one family from another, thus allowing the development of targeted remediation procedures. This approach should not be confused with malware family clustering. In the hereby described task, family labels are available for the known malware binaries (e.g., assigned by previous antivirus checks), as in any training set of supervised learning approaches [8].

Similarity hash functions present distinct drawbacks and outcomes when applied for clustering, detection, and classification. In this work, we explore the three scenarios to investigate their advantages and constraints.

3 Methodology

In this section, we present the adopted scientific methodology on the malware clustering experiments and the dataset considered in them. Whereas we do not claim our scientific methodology to be immediately applied by practitioners, we expect it to be adapted to be applied in conjunction with other existing forensic methodologies (e.g., [40, 3]).

3.1 Dataset Selection

Our goal in this work is to evaluate the application of similarity hashing for clustering malware samples in real-world scenarios. To this end, we searched for a dataset of real malware samples. We considered the dataset characterized in [11] as representative of a real scenario, because it covers a time frame of seven years of in-the-wild malware samples collected from infected user's machines by a security company. This dataset has already been proven to be challenging to other classification tasks [7, 21] such that we believe it might highlight challenges to malware classification using similarity hashing functions as well. This dataset is composed of multiple file types, either executable ones (e.g., EXEs and DLL written in C, CPLs written in Delphi, and code compiled from .Net), or script-based ones (e.g., VBS, Javascript, so on). We selected for our evaluation only the executable binary files present in the dataset, which resulted in a total of 20986 files considered in this study. These binaries are spread over more than 100 distinct family clusters (according to AV's labels), with 53% of them being "Downloaders" or "Password Stealers" variations. Around 40% of all samples are packed (50% of them with UPX and other 50% with multiple, distinct packers). We did not filter obfuscated samples out to demonstrate their impact over analyses in a real inspection case.

To assess the False Positive (FP) rates of the classification procedures, we also applied the tools to a set of legitimate binaries (goodware) that are divided into two classes: (i) 2870 Executable binaries and DLL files retrieved from a fresh Windows 8/x64 installation; and (ii) the 2,935 most downloaded binaries from popular software repositories (e.g., `CNET`) in 2019. The applications were described in a previous study [12]. All goodware files were checked using the Virustotal service to ensure they were downloaded free of bundled malware.

3.2 Clustering Criteria

The considered similarity hashing tools show the similarity between the files `A` and `B` in a pairwise manner, but they do not cluster these files automatically. Thus, we parsed their outputs to create the clusters by ourselves. When clustering the files, it is natural to think about transitivity: if `A` and `B` are similar and `B` and `C` are similar, then `A` and `C` should be similar too. Unfortunately, this might not hold for all files, since they might share distinct code portions. Therefore, we need to choose a criteria to include or not these files into a given cluster. In this work, we decided to include the three files of the aforementioned example in the same cluster whenever the similarity between `A` and `B` and `B` and `C` are greater or equal to the considered threshold for a given experiment.

3.3 AV ground-truth

AntiViruses (AVs) are the most popular defense against malware. Therefore, in this work, we consider AV detection results for ground-truth evaluation experiments. All samples hereby reported as malware were labeled as such by at least one AV engine after the sample's submission to the Virustotal service [70]. In addition to detection rates, we also considered the labels attributed to the samples by the AVs as ground-truth for family clustering experiments. Moreover, we normalized all labels to avoid inconsistency issues [65] whenever the experiments required it. For instance, samples reported as belonging to the `W32/Delf.A` and `W32/Delf.B` families were assigned to the same cluster. In the case of "Generic" labels, we assigned the samples to a new cluster to reflect the fact that AV analysts will have to manually inspect the "unknown" samples. Depending on the experiment goal, singleton clusters are discarded. All labels were retrieved in Dec/2019 and attempts to reproduce our results should refer to these scans, as AV labels may change over time [13].

3.4 The Ideal AV Model

All experiments are described in this paper as if they were applied in the light of an ideal, Similarity Hashing-based AV Engine (SHAVE). The reliance on similarity hashing functions allows us not only to match identical binaries (100% similarity), as in typical signature-based detection schemes, but also extend AV detection capabilities to detect samples based on partial matches. We restricted our threat model to signature-based AVs to not bias our results with influences of other detection mechanisms, such as heuristic detectors. The ideal AV engine is structured in a client-server architecture. The client running on the user's endpoints triggers inspection procedures according to user demands and uploads the digest of the suspicious file to the server running on the cloud. The server is responsible for effectively implementing the detection logic, which, in the case of this work, relies on similarity metrics. The server compares the suspicious digest against a previously built database of similarity digests having multiple known threats and reports the similarity score. The suspicious file is considered as malicious or not according to the established threshold. This can be configured either on the client or server-side. The major advantage of the considered approach is that the AV only uploads a digest to the cloud instead of an entire file, unlike some cloud-based AV solutions. This allows preserving user's privacy and saving network traffic. The centralized database also allows instant malware definition updates, as these do not need to be distributed to endpoints via the Internet. Therefore, each scan request is always performed against the latest version of the malicious digest database. Whereas the experiment presented in Section 4 justify SHAVE's viability, we do not expect it to be adopted isolated as in our experiments, but in conjunction with traditional AV solutions.

4 Experiments & Results

In this section, we present and discuss the experiments performed to evaluate the feasibility of leveraging similarity hashing function for multiple, distinct malware-related tasks. More precisely, we first describe the characteristics presented by similarity hashing functions when characterizing a real dataset of malware samples. We further evaluate whether this type of function allows one to cluster unknown binaries to separate them into malicious and goodware samples. We finally evaluate whether one can leverage this approach to identify distinct malware families.

4.1 Characterizing a Real Dataset: Malware Family Clustering

An immediate application for similarity hashing functions is to characterize an unknown dataset regarding the number of malware families and their sizes, which allows analysts to make inferences about the given scenario, such as how active and diverse it is. Dataset characterization, however, is not a straightforward task, but depends on multiple factors. In this work, we are interested in shedding light on some of them. In particular, we considered two main affecting factors: (i) the used similarity hash function; and (ii) the similarity threshold value.

The used similarity hash function directly affects the final result as each function is developed using different techniques (see Section 2) and thus output a distinct similarity score. Pagani et al. [56] identified that the `sdhash` function outperforms the popular `ssdeep` when applied to their dataset of collected samples. We here applied these same functions classes[1] to a dataset of malware samples collected in-the-wild to evaluate whether this conclusion holds in broader scopes.

The threshold value affects the result because it conceptually defines when a sample is considered similar to another. Failures in adopting a proper threshold value might lead to False Positives (FPs) or False Negatives (FNs) in the case of malware detectors and/or classifiers. In the following, we discuss the extent of the influence of the threshold value in the overall dataset characterization.

The similarity hash functions. We evaluated the impact of distinct similarity hash functions by applying them to the dataset described in Section 3. Figure 1 and Figure 2 show, respectively, the relative number of clustered samples and the total number of identified clusters by the two similarity hash functions according to the threshold variation (from 0 to 100). Figure 1 shows that the tighter the considered threshold, the fewer samples are identified as similar. `J-sdhash` slightly outperforms `ssdeep` for all thresholds, which confirms Pagani's results. Figure 2 clarifies that, when using laxer thresholds, the similarity hashing functions not only add more samples to already-existing clusters (which could be due to False Positives), but they indeed generate more clusters to host samples whose similarity was not identified using distinct thresholds. From an incident response perspective, it brings two significant advantages: (i) the more samples are identified as similar, the faster they can be handled by a uniform procedure; and (ii) having more specific clusters allows more fine-grained incident response.

Finding #1: *The J-SDHASH tool covers more samples and generates more clusters than the SSDEEP tool, thus easing malware triaging.*

The threshold. The threshold value is key for a good classification analysis, but it seems there is no consensus in the literature about which is the ideal value. One can find related work suggesting values of 5 [62], 20 [43], and 90 [27]. In practice, researchers end up adopting ad-hoc threshold

[1]We used `J-sdhash` instead of the original `sdhash` version

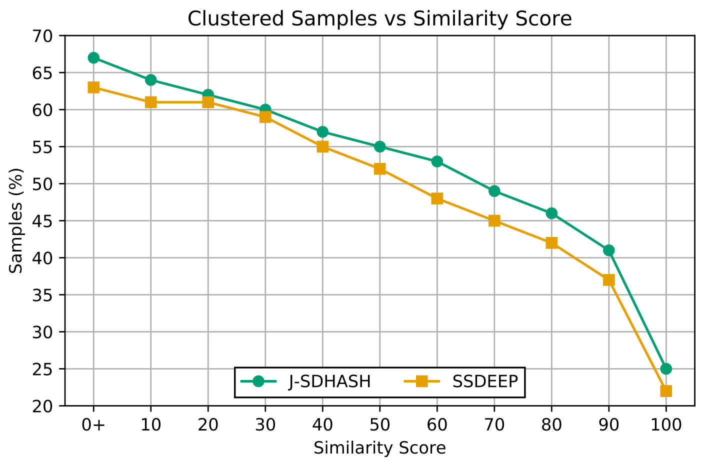

Figure 1: Relative Number of Clustered Samples per Similarity Threshold for the two similarity hashing functions.

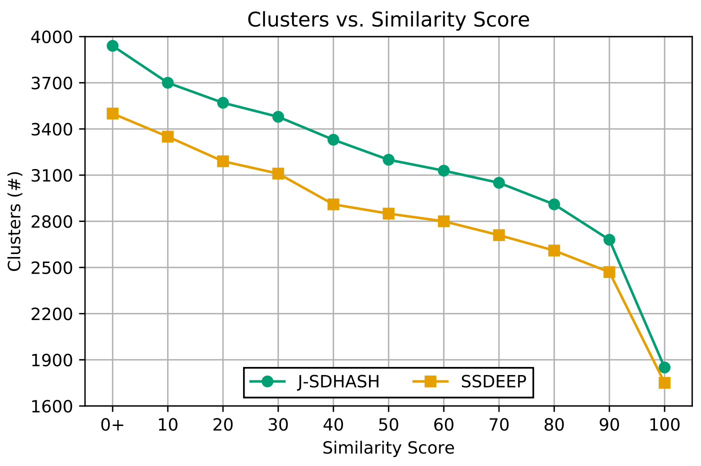

Figure 2: Absolute Number of Clusters per Similarity Threshold for the two similarity hashing functions.

values among the ones considered in our experiments, since there is currently no guideline for the threshold definition. Understanding and defining threshold criteria is an important open research question because the variation on the number of clusters and clustered samples in some scenarios might be expressive enough to lead to contradictory conclusions. For instance, in the considered dataset, on the one hand, a lax threshold of 50% or less (Figure 1) would support the conclusion that more than 50% of the dataset is composed of malware variants (a scenario with few but very active attackers, as discussed below). On the other hand, a strict threshold value of 100% (Figure 1) would support the conclusion that less than a third of the dataset is composed of similar samples (a scenario with more but less active attackers, as discussed below). Therefore, researchers should (i) define a clear threshold for their experiments instead of adopting default settings; and (ii) avoid comparing datasets using distinct thresholds. In the following, we discuss additional criteria for threshold definition.

Finding #2: *Distinct threshold values might lead to completely opposite conclusions about the characteristics of the same dataset.*

To better understand the impact of multiple threshold values, we took a look at the cluster size distribution in addition to the number of clustered samples. The cluster size distribution allows understanding and inferring multiple characteristics of the observed dataset. One can, for instance, leverage the number of distinct identified malware families as a proxy for the number of attackers distributing malware samples and/or attack sources: the greater the number of distinct clusters, the greater the chance that the associated malware samples were generated by distinct attackers. Alternatively, one could use the sizes of the clusters as a proxy for characterizing how active the attackers are: the greater the number of samples in a cluster, the greater the chance that the attacker remained active for a long time generating new malware variants.

We evaluated how clusters are distributed for the distinct threshold values. Figure 3 illustrates the cluster size distribution when the 50% threshold is applied[2]. We notice that small clusters (e.g., sized 2, with most samples being associated with at most one other sample) are more prevalent than large clusters, thus indicating a variety of malware threats. This characteristic also holds for all other threshold values we investigated. It was also reported in previous studies [27], thus indicating it is a typical characteristic of real malware datasets and that it can be observed regardless of the considered threshold value.

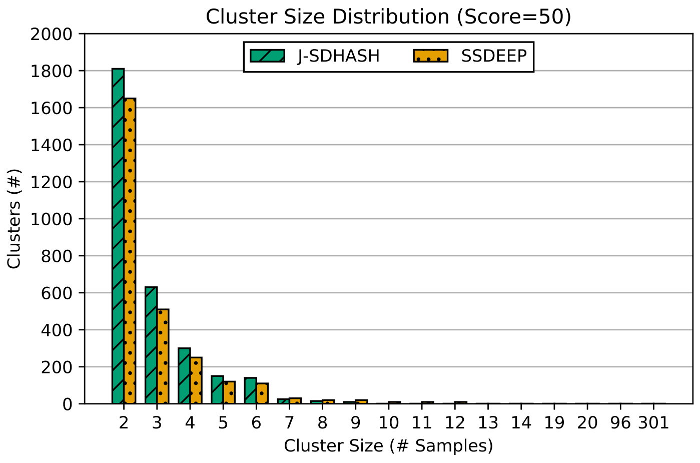

Figure 3: **Cluster Distribution for the 50% threshold.** Smaller clusters are more prevalent than larger ones, thus incidating sample's diversity.

Our experiments revealed that the threshold change affects both the maximum number of samples attributed to a single cluster as well as the size of the largest identified cluster. These factors are associated, since the threshold change "moves" samples back and forth from new clusters to already-existing ones. For instance, our experiments show that the clusters sized 2 are the prevalent ones for all threshold values, as shown in Figure 4, even though their number is reduced for tighter threshold values. This happens because the use of tighter thresholds implies that some samples will

[2]chosen randomly as a single example due to space constraints

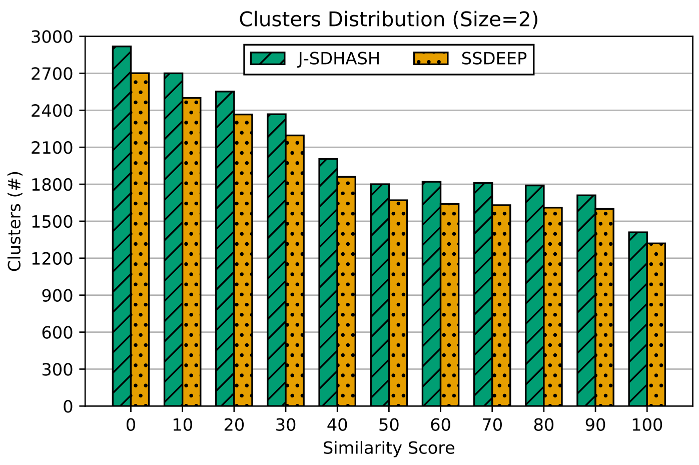

Figure 4: **Clusters Sized 2** are prevalent for all threshold values, although reduced in number for increased threshold values.

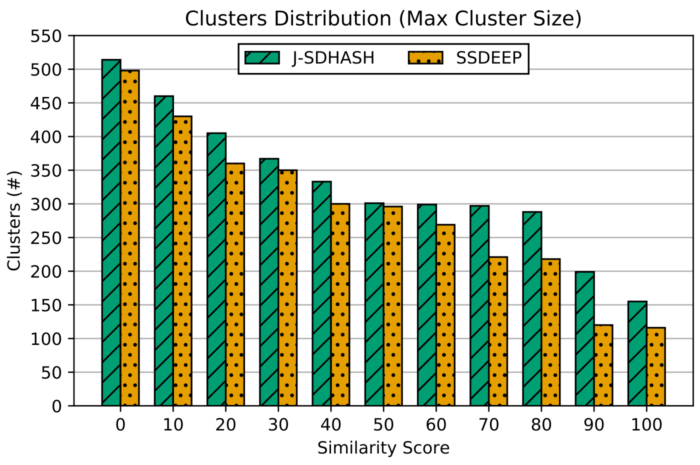

Figure 5: **Maximum Cluster Size.** The tighter the threshold value, less large clusters and more specialized ones are generated.

not be considered as enough similar anymore, thus being removed from any cluster. This suggests that laxer thresholds are more suitable for tasks that require maximizing the sample's coverage by the similarity hash function.

Our experiments also show that the size of the largest clusters also change when tighter threshold are considered, as shown in Figure 5. This happens because tighter thresholds tend to divide single, large clusters into multiple, smaller ones, according to the samples that best match themselves. This suggests that tighter thresholds are more suitable for tasks that require sample's clusters to be highly specialized.

More specifically, two classes of analysis tasks require distinct cluster characteristics to operate with. When triaging malware, it is desired to have the greatest coverage possible to minimize the processing costs; When applying remediation procedures, it is desired to have the most similar clusters possible to apply the correct, specialized vaccines to them. Therefore, we conclude that there is not a single threshold value that applies to all tasks, but that smaller threshold values should be used for malware triaging and higher thresholds for remediation procedures. For both cases, we observed that the `J-sdhash` function outperformed the `ssdeep` function for all threshold values. In the following subsections, we dig deeper into these tasks.

Finding #3: *Smaller thresholds are more interesting for malware triaging and larger thresholds are more interesting for remediation.*

4.2 Detecting New Threats: Malware Detection

A common way to detect malware is to check whether an unknown binary clusters to a set of known malware binaries, according to the considered clustering criteria. We here discuss how similarity hashing functions can assist in this task. For all the following presented experiments, the modified `sdhash` function (`J-sdhash`) was considered, as it was the one that produced the greatest variety of clusters in the experiments presented in the previous section.

AV Detection Rates Increase using Similarity Information. Antiviruses (AVs) often perform their clustering tasks using deterministic signatures, which is reasonably effective (95% of all samples in the real dataset were detected by at least one AV). Our hypothesis is that similarity hashing functions could also be used for this task and perform better. In the following, we present an evaluation of how AV's detection rates could be individually increased by considering binary's structural similarity information. We evaluated the detection rate when looking at the similarity between a known malware sample and a suspicious sample. More specifically, we searched for a tuple of samples (A,B) that either A or B was detected by a given AV solution (but not both) and the similarity between A and B is greater than a predefined threshold. All experiments were performed using the same aforementioned dataset and all AV solutions available in the VirusTotal service were considered.

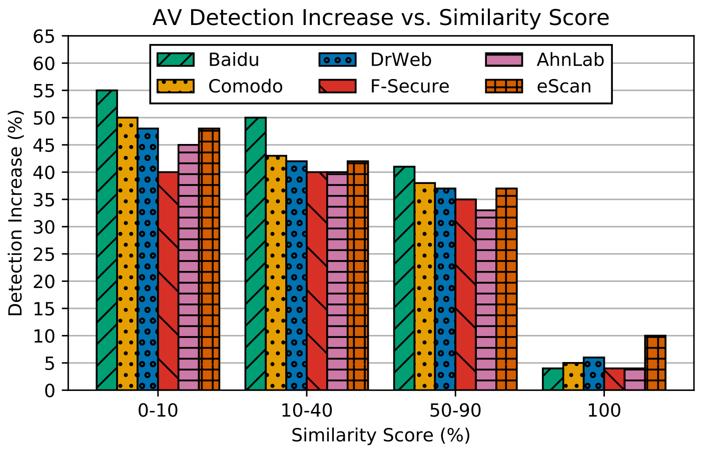

Figure 6: **AV Detection Increase when considering binary similarity.** Similarity effectively increases AV detection.

Figure 6 shows the detection rate increase according to the multiple thresholds for the most affected AVs–i.e., the AVs which would most increase their detection rates if similarity hashing capabilities were added to them. As expected, the laxer the threshold, the more samples are considered similar to known malware and thus the greater the detection rate increase. We notice that even when a tight threshold of 100% is considered (which requires the known malware and the unknown payload to be 100% compatible–even though not 100% equal), the detection rate keeps increasing, thus showing that there is room for improvements in AV detection strategies and capabilities (such as by using similarity hashing functions).

Finding #4: *The use of similarity hashing functions by AV increases their detection capabilities in practice.*

Whereas increasing the detection rate via the application of similarity hashing functions is possible, and even very significant in the presented cases, it is important to highlight that distinct AVs will be affected distinctly, according to their engine's weaknesses and capabilities. Figure 7 presents a landscape of the detection rate increase for the multiple AVs present in the Virustotal service. We notice that all AVs were positively affected, even though by a small factor

The case for False Positives. We are not aware of AV solutions using similarity hashing functions in the same way as proposed in this work, although our results suggest that adopting this approach is a promising way of increasing malware detection. This fact raised the concern if any limitations were preventing AVs from adopting this type of operation mode. We hypothesized that a plausible explanation could be an increase in the False Positive (FP) rates when similarity scores were considered–i.e. if the laxer thresholds were resulting in goodware samples being considered similar to malware samples. To evaluate that possibility, we repeated the same experiments as described above but now considering the two datasets of goodware samples previously described in Section 3.

Table 2 shows the FP rate (regarding the detection by any Virustotal's AV) for distinct threshold values in the two datasets. Around one out of four samples result on a FP when using the minimum threshold value of similarity supported (threshold $> 0,1\%$). Increasing the threshold considerably decreases the FP rate. A threshold of 50% decreases the FP rate to 1%. If we consider that a perfect compatibility score (100% threshold) between known malware and suspicious payloads is required to classify a payload as malicious, we observe an almost negligible FP rate while still allowing AV detection rates to increase, as previously shown. Therefore, our findings show that FP is not a limitation for the application of similarity scores to increase AV's malware detection.

Finding #5: *A threshold of 50% is the best trade-off between TP and FNs when using a similarity hashing function to assist AV detections.*

Similarity Hashing Functions Performance. The major advantage of the signature-based approaches adopted by many AVs is that they are very fast. Similarity hashing functions are more complex than traditional hash functions–`J-sdhash` even requires the application of a traditional hash function as part of its operation (see Section 2), thus imposing a greater processing load. We, therefore, evaluated whether the greater performance impact could be a limitation for their application in real-world scenarios. For such, we measured the throughput of distinct similarity and traditional hash functions when inputted with real malware samples. All experiments were performed on a 64GB, Intel Xeon E5-2620 2.00GHz CPU running Ubuntu 18.04 LTS.

Table 3 and Table 4 show the average throughput (hashes per second) while hashing the entire malware dataset described in Section 3 using distinct hash functions, considering, respectively, the CPU Time (actual processing time) and the Wall Time (total time spent). We notice that for all cases the throughput reported using CPU Time is an order of magnitude greater than Wall Time, thus showing that the time taken to load the hash database to memory is not negligible (which is key for an AV).

Finding #6: *Similarity hashing-based AVs should not neglect the cost of loading the signature database in memory to operate even though they might reduce the database size by condensing similar file's signatures.*

We observed that all the traditional hash functions presented greater throughput than the similarity hash functions, which shows that the distinct performance rates are due to the distinct nature of the functions rather than particular implementation decisions. We also observed that the previously shown increase in `J-sdhash`'s similarity results in comparison to `ssdeep`'s is obtained via a decrease in its performance. This result suggests that whereas the `J-sdhash` function should be selected in unconstrained scenarios (e.g., offline analyses) due to its higher classification capabilities (shown in the previous subsection), the `ssdeep` function is a better option for performance-limited scenarios (e.g., real-time operation), since it provides a ≈ 4 times greater throughput without losing ≈ 4 times classification power.

Finding #7: *Whereas J-sdhash is a better choice for unconstrained scenarios, ssdeep is a better choice for performance-limited scenarios.*

Similarity hashing performance and AV databases. In addition to performance penalties when hashing individual files, similarity hashing functions also do not scale well naturally (i.e., without a proper approach/protocol), which might be prohibitive for an AV company trying to establish a database of known malware hashes. Figure 8 shows the spent time to naively match a file against the totality of a growing database, up to the limit of the entire malware dataset. This simulates the addition of newly-discovered samples to an AV company database, as AV companies often perform in traditional hash-based procedures. This kind of growth might seems prohibitive at a first glance.

The viability of leveraging similarity hashing functions in real-world cases is only highlighted when we consider that

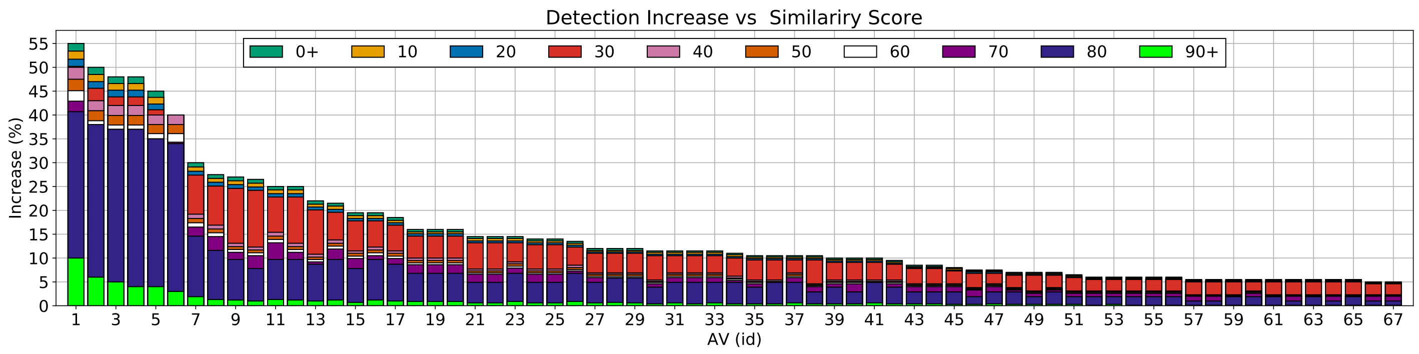

Figure 7: **Overall AV Comparison.** Distinct AVs increase their detection rates differently.

Table 2: **False Positive Analysis.** Few goodware samples are mistakenly flagged as malicious.

	Win/Sys32			**Repositories**		
Threshold	0+%	50+%	100%	0+%	50+%	100%
FPs	27.30%	1.00%	0.05%	25.00%	1.00%	0.05%

Table 3: **Hash Functions Throughput Comparison.** Hashes per second considering CPU processing time.

Function	**Throughput**	**Function**	**Throughput**
J-sdhash	948	SHA256	5651
ssdeep	2475	MD5	9777
SHA512	5146	SHA1	11110

Table 4: **Hash Functions Throughput Comparison.** Hashes per second considering Wall time.

Function	**Throughput**	**Function**	**Throughput**
J-sdhash	82	SHA256	613
ssdeep	369	MD5	727
SHA512	413	SHA1	787

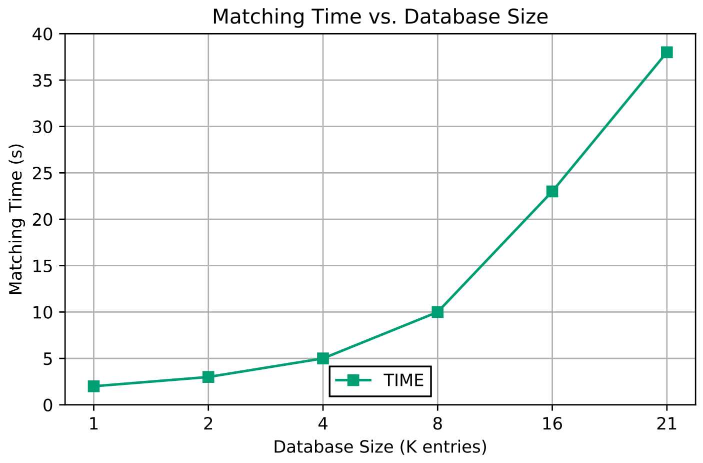

Figure 8: **Matching Time.** The matching time of similarity hashing function grows exponentially.

they do not need to perform the same number of comparisons as traditional hash functions to produce the same result. Whereas typical hash functions would require the comparison of all thousand files present in the dataset to check whether any of them is malicious, a similarity hash function would require checking only the centroids of the corresponding clusters, thus reducing the problem space by a magnitude order. This is possible because any sample in the dataset is similar at least to its cluster centroid by construction. Therefore, to a similarity hash-based AV to be practical in terms of performance, a good clustering approach at the server-side is required.

There are distinct approaches to accelerate the digest search for large databases. The strategies might cover databases in up to $O(n \cdot log(n))$ or $O(n)$ comparisons, depending on the chosen strategy and the problem constraints (more information about these strategies can be found in [48]). However, a common drawback of all these approaches is that they consider an already-built, fixed-size database, which is not the case of AV companies, that build their databases as soon as new samples are being discovered.

To better understand the impact of centroids selection on AV's performance, we evaluated how fast a reference dataset (the AV database) can be covered in practice, again simulating an AV company creating a digest database, adding more samples to it, and querying it for detection. For such, we considered a DBSCAN-like algorithm that takes a new sample (randomly chosen for the sake of the experiment) and clusters it to an existing cluster in the AV company's database if the score is above the predefined threshold, or creates a new cluster if not. This type of approach has been previously used along with other similarity hash functions [32, 71] and we are here extending it to operate along with `J-sdhash`. In a real operation, this procedure is indefinitely repeated for every new incoming sample. In our experiment, the procedure repeats until all clusters (with regards to the non-probabilistic ground-truth) are formed. We considered 100 repetitions of this algorithm while initializing it in random states.

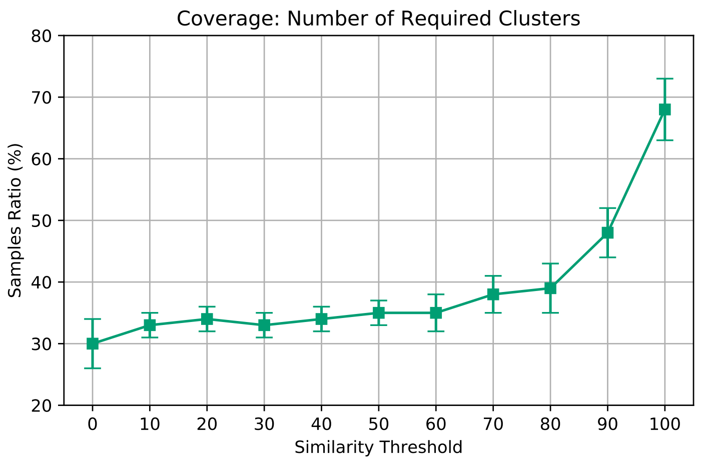

Figure 9: **Clustering:** Number of Samples.

The average results presented in Figure 9 show that the performance gains are greater as the laxer the thresholds. When a 50% threshold value is considered, the match against ≈ a third of the dataset is enough to cover all samples. In other words, if an AV company had built its database using ≈ a third of the samples we considered in the experiment, it would be able to detect the remaining two-thirds due to their similarity with the ≈ one-third of centroids. When the threshold value is increased to its limit (100%), around 70% of the samples are required to achieve full coverage. In other words, the AV company should have access to 70% of the samples to detect the remaining 30% of the samples. This result is explained by the fact that when a strict match is considered, the result is bounded by the great number of clusters sized two, as shown in the previous sections, thus requiring more checks to cover all clusters. Therefore, the larger the reference clusters, the better the performance of the AV scanning schema.

Finding #8: *A good threshold selection allows an AV database that covers the full reference dataset to be developed with only a third of the reference samples.*

4.3 Response & Remediation: Malware Family Classification

In addition to detecting malware, AVs also try to classify them in families. This is essential to label the threats and allows incident response procedures and remediation (in the ideal scenario). As for malware detection, previously presented, we also hypothesized that similarity hashing functions could assist AVs in this task. In the following, we present our findings.

AV Ground-truth is hard. The first step to evaluate whether similarity hashing functions are good for malware family clustering is to adopt a ground-truth, i.e., to identify which families malware samples actually belong to. In our experiments, we used the most adopted strategy in the literature: the reliance on AVs labels. We consider this choice reasonable since although there are alternatives for malware labeling (see Section 7), AVs are often the only immediately available tool for most analysts. Notice that having to rely on a third party to label similar samples is the case for most real-world datasets (including ours) where there is no previous information about the sample's similarity.

To evaluate AV's agreement with the similarity hash functions results, we considered two cases: individual AV's classification capabilities, and the classification capabilities by an AV committee. For such, we modeled two hypothetical AVs from the labels assigned by VirusTotal's AV engines. The first AV we modeled is an "ideal" AV that confirms that two samples are really similar whenever two labels assigned by any VirusTotal AVs agree (even if the labels are provided by distinct AVs); The second AV we modeled considers the case of an "average" AV that confirms that the samples are similar whenever two labels assigned by the same VirusTotal AV agree (for the same AVs).

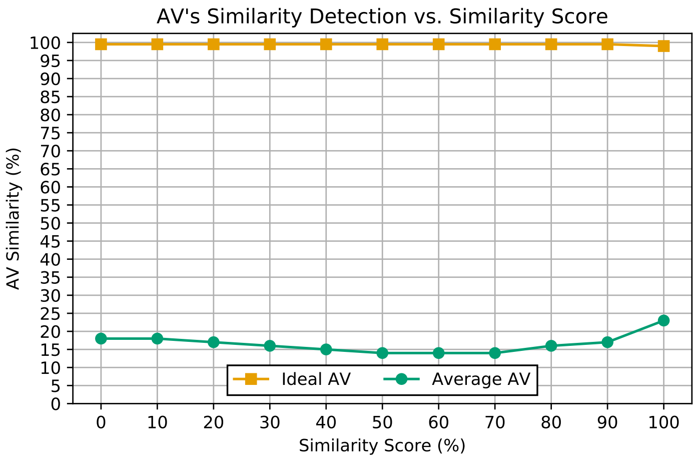

Figure 10: **AV ground-truth.** Samples reported similar by an "ideal" and an "average" AV.

Our first finding was that relying on individual AVs for establishing malware families ground-truth is a challenging task since the labels hardly ever agree. Figure 10 shows the agreement between the samples identified as similar by `J-sdhash` and the two hypothetical AVs that we modeled. The fact that most of the samples considered similar by the similarity hash functions are also reported similar by the "ideal" AV confirms that the similarity hash function is able to effectively detect similar samples. However, the low "average" score indicates that few AVs are able to individually detect similar samples, which makes their use as ground-truth labelers hard. In other words, the difference between the "ideal" and the "average" AVs shows that the distinct approaches adopted by each AV solution are effective to detect similar malware samples when combined but not when applied individually.

Finding #9: *An AV committee is able to confirm the similarity results presented by the functions, but individual AVs are not.*

A significant factor limiting the performance of the average AV is that AVs often generate "generic" labels. When it happens, one has to either try to cluster the generic sample anyway or generate a new cluster for the unknown sample. In both cases, the correctness of the clustering process is affected, either due to False Positives or False Negatives. Using a committee mitigates this problem by diluting the responsibility on the decision procedure, since it is very unlikely that all AVs will present generic labels at the same time. Thus, relying on the AVs diversity mitigates the generic labeling problem.

The average result is highlighted when we observe the differences between individual AVs when clustering the samples. Figure 11 shows the AVs that clustered most malware samples into families and the number of clusters identified by each one of these solutions. We notice that each solution identified a distinct number of clusters, thus showing that they rely on distinct criteria for the sample's classification.

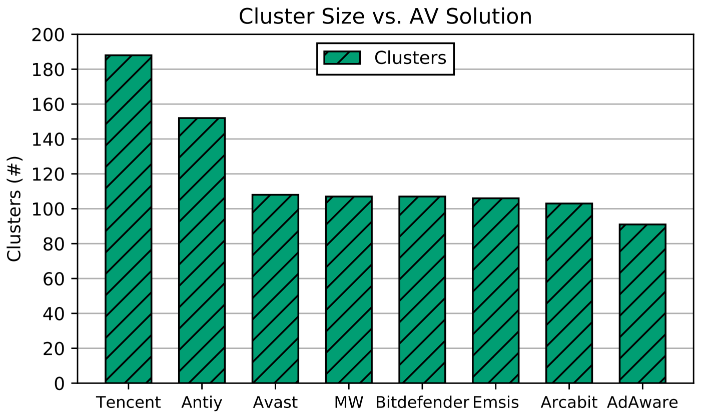

Figure 11: **Number of clusters generated by different AV solutions.** Each AV solution generates a distinct number of clusters.

Figure 12 shows the cluster size distribution for the AV that most clustered samples. The observed distribution is clearly different from the distributions obtained using the similarity hash functions (shown in Section 4.1, with a huge prevalence of clusters sized 2 and without a thousand-sized cluster), even when considering the multiple evaluated threshold values. This result also suggests that the AV relies on a distinct method for malware clustering than the use of a similarity hash function.

Finding #10: *The criteria used by AVs to classify samples as similar is completely different from the used by the similarity hashing functions.*

The Root of AV labels. Given the identified heterogeneous results, we started investigating which types of clustering mechanisms were used by the AV solutions. Table 5 shows the most popular labels assigned to the largest clusters. We discovered that AVs have been labeling samples according to heuristics, which flag samples as similar when they are all matched using the same rule, regardless of their structural similarity. We can observe, for instance, that many samples are classified in the same "AutoIt" family, which means that all malware samples were created using this same framework. This label, however, does not provide any information about the structural similarity between these samples. Other samples were grouped in the "Themida" cluster, which means that they were all packed using this same solution. Unfortunately, this label cannot say much about the similarity or behavior of the embedded payloads.

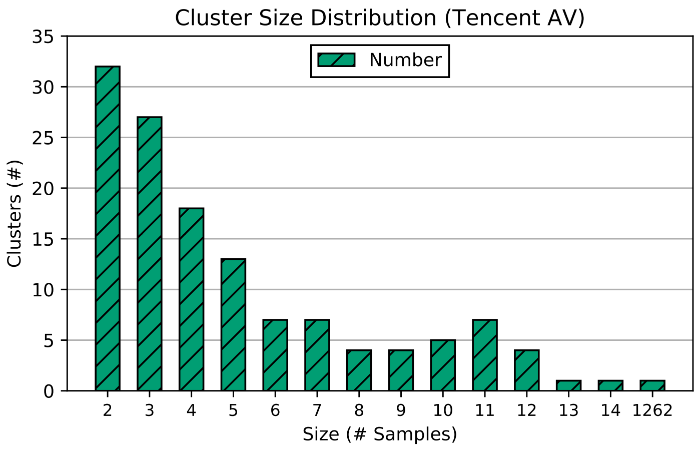

Figure 12: **Cluster histogram of Tencent AV solution.** The distribution is clearly different from the ones generated from similarity hashing functions.

Table 5: **Tencent Cluster's Labels.** The AV clusters samples based on heuristics and not on binary similarity.

Cluster	Samples (#)	Cluster	Samples (#)
Delf	437	AutoIt	385
Proxy	12	Heur	10
Themida	4	Rogue	2

Finding #11: *AVs flag samples as similar when they present the same file packaging and/or when are packed with the same solution, regardless of their content.*

The Impact of Packing. The previous results show that AV labels are based on binary packing. This might significantly affect detection and labeling since malware often uses packers to hide their payloads from analysts and analysis tools. Given the potential impact of packing over malware classification, we decided to investigate what happens when similarity hashing functions are applied to packed samples. Ideally, we would like to check all packed samples, but there is no automatic unpacking solution for all type of packers identified in the samples of our dataset. Thus, we limited our analysis to UPX [53], a popular open-source packer, since it packed 50% of all packed samples of our dataset.

Figure 13 shows the comparison on the relative number of unpacked and UPX-packed malware samples clustered by the `sdhash` function. We first notice that the unpacked versions are more clustered than the packed versions, which shows that the packing tool can hide some sample's common characteristics. The tighter the selected threshold, the greater the relative difference in the number of packed and

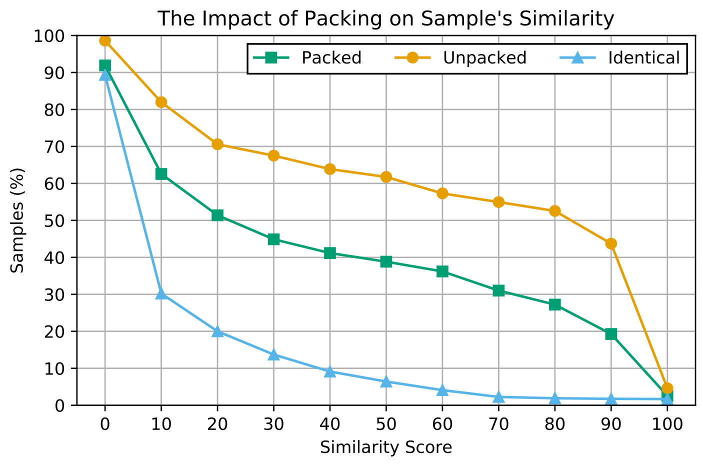

Figure 13: **The impact of UPX packing.** Packing reduces sample's similarity scores.

unpacked samples, as the impact of the few common characteristics identified by the similarity hashing functions is less weighted. Nevertheless, packing similar samples does not completely eliminate the capabilities of hashing functions to cluster them. This happens because similar samples packed with the same solution result in similarly packed binaries, which are then clustered. The "identical" curve shows the matching rate of the same samples in their packed and unpacked versions (discarding non-original samples similar to them). We notice that the packer significantly changes the binary structure, such that most packed samples are very distinct from their original versions. This allows us to conclude that the samples that were clustered by the similarity hashing functions in their packed versions are not similar with their own unpacked versions, but to multiple packed variants of themselves distributed by the attackers.

Finding #12: *Malware packing with UPX significantly reduces clustering capabilities but does not completely eliminate its application to same-packer variants.*

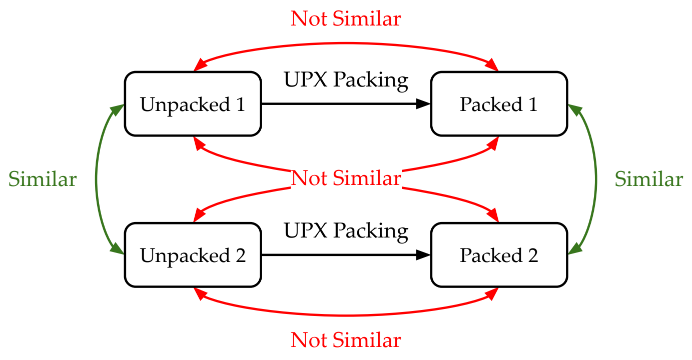

Figure 14: **Average Packed Sample's Similarity Scheme.** Cross-comparisons should be avoided.

Figure 14 summarizes the similarity scheme we identified for the majority of the inspected samples. Notice that this result does not imply that all packers will present this same characteristic when generating malware variants. Packers that perform distinct transformations for each run (e.g., crypters that use distinct keys) might generate distinct variants from the same original code.

AVs Agreement. A drawback resulting from the differences between our approach for similarity hashing clustering for family identification and the current methods leveraged by AVs to label samples is that these two outcomes cannot be directly compared. If they are, a low agreement between the samples considered similar by the hash function and the AV is observed. In this work, we understand the label agreement of a given cluster as the ratio of the samples in the cluster that present the same label. For instance, if all samples in a given cluster defined according to the similarity hash function scores present the same AV labels, their agreement is 100%. If 2 out of 3 samples in the same cluster present the same AV label, their agreement is 66%. If all samples present distinct AV labels, their agreement is zero. Ideally, we would like that the similarity hashing functions and the AVs presented perfect cluster and label agreement rates (100% of agreement for the labels of 100% of all clusters).

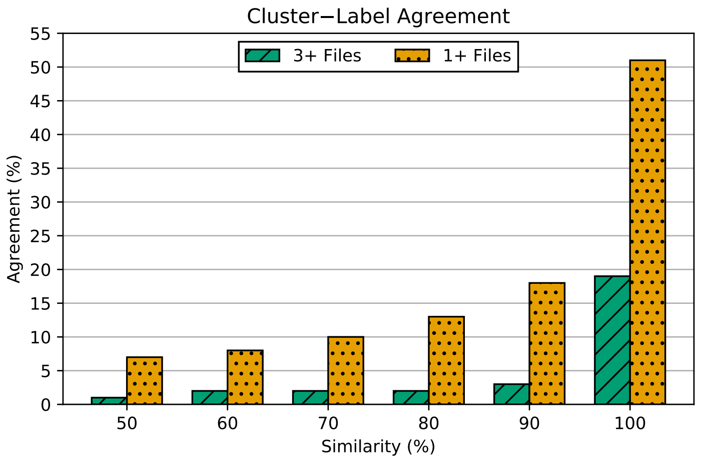

Figure 15: **Agreement between the clusters generated by the AV and by `J-sdhash`.** We notice that most samples clustered by `J-sdhash` present divergent AV labels.

Figure 15 shows the fraction of clusters that presented a perfect label agreement rate for distinct thresholds. For a strict threshold of 100%, if we consider all samples in the dataset, the average agreement ratio is 50%, i.e., 50% of all clusters agree 100% in their sample's labels. Whereas this is not a very high value, it is still surprising, since it seems to contradict the previously presented difference on the cluster distribution for distinct thresholds (Figure 3). We then realized that this result was biased by the large number of small clusters, with two samples only (Figure 4), which tend to agree more due to their low intra-cluster diversity. The agreement ratio of clusters with three or more samples is around only 20%, thus showing that using AV labels as ground-truth for cluster labeling is a challenging task. This same phenomenon was observed when we varied the threshold.

Finding #13: *The direct application of AV labels over similarity hashing functions-generated clusters leads to low intra-cluster label agreement rates.*

On the positive side, the tighter the threshold, the greater the label agreement, thus showing that the clusters tend to be more specific, thus more similar, in these cases. This confirms our finding 2 about the best suitability of tighter thresholds for remediation procedures rather than triaging.
Selecting individual AVs. Despite the presented limitations, we cannot completely discard individual AVs as labeling tools for two reasons: (i) they are still the only solution available to label unknown payloads in many scenarios; and (ii) many experiments can only be validated by using a consistent evaluation tool. For instance, similarity hashing function's classification improvements can only be properly validated by using the same AV as ground-truth, otherwise, one cannot ensure that a better classification result is due to the proposed function improvement rather due to the reported matching of the file against a distinct AV part of a committee.

Face to this scenario, it is essential to develop criteria for selecting an AV for the experiment development. Since our ultimate goal is to evaluate the similarity hashing function, we proposing that the AV that most resembles the similarity hashing function results should be selected. Thus, we propose that we should consider the usual metrics of accuracy, precision, recall, and F1 score for comparing their outputs. In our context, a TP occurs when the AV assigns the same labels to two samples that the similarity hashing function considered similar. A TN occurs when the AV assigns distinct labels to two samples that the similarity hashing function considered non-similar. A FP occurs when the AV assigns the same labels to two samples that the similarity hashing function considered non-similar. A FN occurs when the AV assigns distinct labels to two samples that the similarity hashing function considered similar.

The first hypothesis that comes up in everybody's minds for selecting an AV is that the AV which detects most samples is also the one that best clusters them. However, this might not hold for all cases. In our experiments, AVG was the AV that most detected samples, but the use of its labels for clustering purposes resulted in poor accuracy rates, as shown in Figure 16.

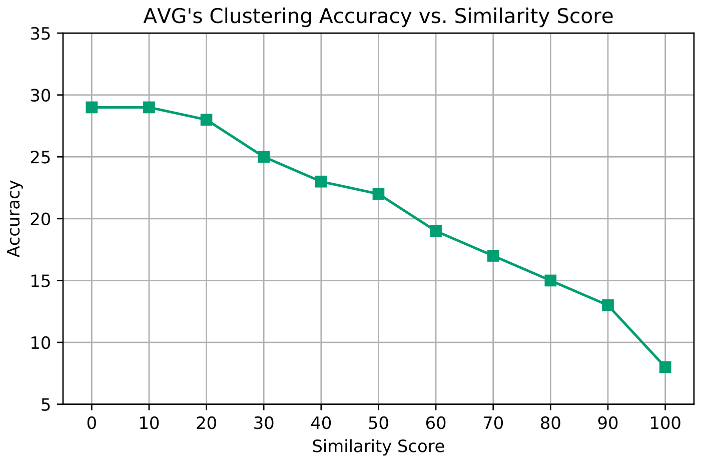

Figure 16: **AVG Clustering Accuracy.** Despite presenting the greatest detection rates among all AVs, AVG was not the AV that most/best clustered samples.

Finding #14: *The AV that detects most samples is not necessarily the one that best clusters them.*

To identify the AV that actually best fits the similarity hashing function, we repeated the experiment to compare the metrics for all VirusTotal's AVs. Figures 17, 18, 19, and 20 show, respectively, accuracy, precision, recall, and F1-scores for the AVs that best performed in our evaluations. The NANO AV was the one that achieved the best scores for all metrics.

We notice that the accuracy values grow in the 0-30% range, which might be due to TP or TN increase. However, AV labels seem to impose an upper bound after the 30% value, which prevents the metric from growing significantly. Precision values confirm that TP also increases in this range, which is explained by the similarity hashing function generating clusters of more similar samples as tighter the threshold, as previously presented. However, this same characteristic make recall values to lower as the threshold increases, because the similarity hashing function stops clustering together samples that the AV labels state as similar. As previously shown, this behavior is due to the broad heuristic and generic labels assigned by the AVs to cover samples packed in the same format despite its content. The F1-score metric weights these findings and shows that the best results are achieved for intermediate threshold values. We notice that the F1-score for the 100% threshold is low because whereas it correctly labels very similar samples, it leaves all the generic, heuristic-labeled samples out of the clusters.

In other words, we observe that, despite all challenges, some AVs can present a reasonable trade-off between TPs and FPs, depending on the considered threshold values. The balance is completely tied to the way that the AV labels the samples. For instance, more conservative AVs will cluster fewer samples as similar than similarity hash functions, thus raising the number of false negatives and limiting the precision rate. Thus, in practice, the AV drawbacks when labeling samples turns into an upper bound for the performance rates.

Finding #15: *Choosing the AV that best fit similarity hashing function results is the best choice for individual ground-truth, but this might impose an upper bound limitation to the evaluation metrics.*

The Root of Similarity. Once we identified the nature of the labels assigned by the AVs and the AV solution that best fits the similarity hashing function's behavior, we started investigating the nature of the similarity matches at the byte-wise level. This task requires us to split the files into parts to identify which one of them contributes most to the similarity score. A natural choice for such splitting is to consider the natural organization of executable binary files in sections. In this study, we considered Windows malware samples, that

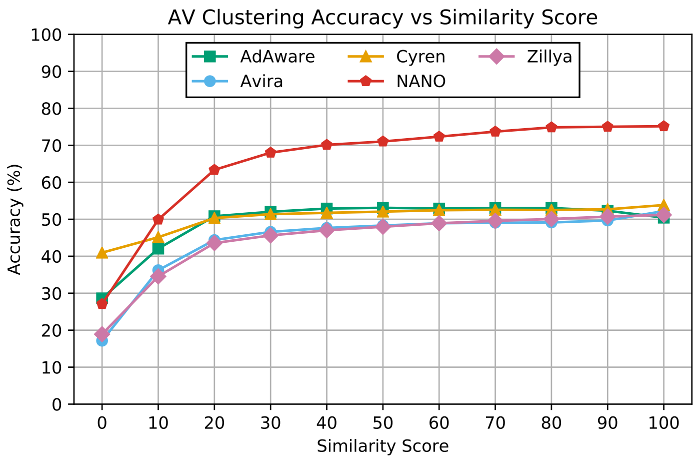

Figure 17: **Whole Binary Accuracy**

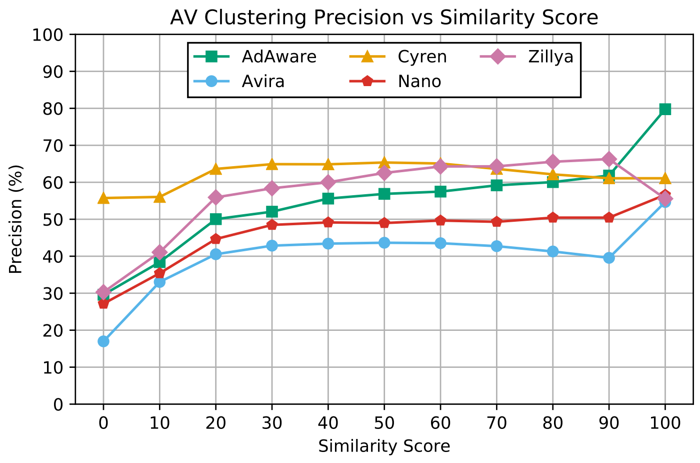

Figure 18: **Whole Binary Precision**

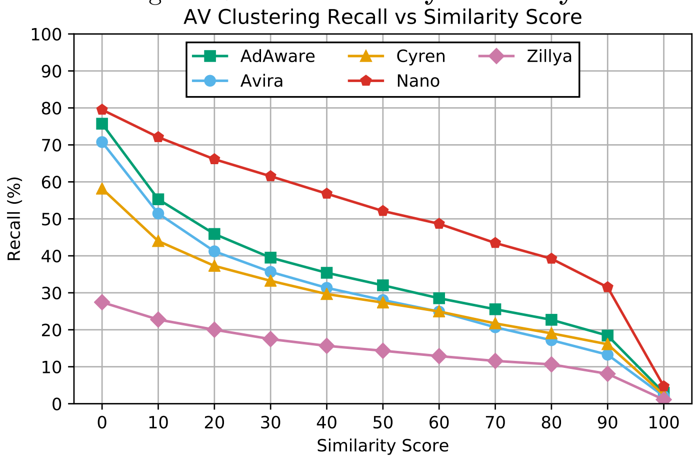

Figure 19: **Whole Binary Recall**

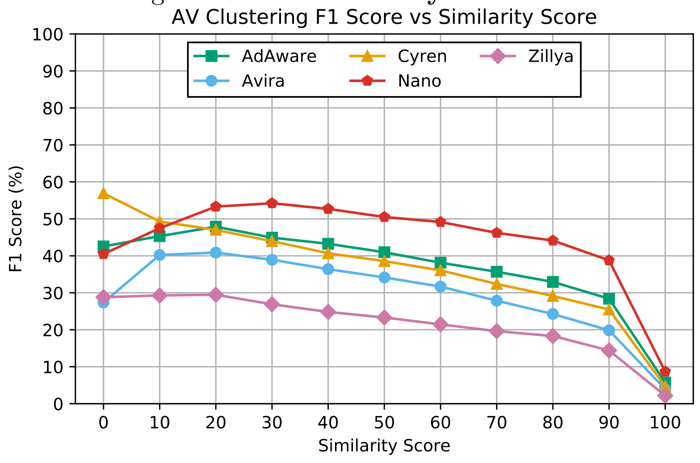

Figure 20: **Whole Binary F1 Score**

are distributed via the Portable Executable (PE) [58] file format. The PE format is very flexible and does not define fixed section names and permissions, such each sample might present its own combination. To triage all possibilities, we started by investigating all sections presented by the binaries, as summarized in Table 6.

We discovered that the diversity in binary sections is due to the attacker's implementation choices. Thus, each section represents a distinct implementation decision and has a distinct goal. For instance, whereas "`text`" sections often implement malicious actions by themselves (instructions), data sections are often used to carry payloads injected by third parties. Similarly, sections such as "`UPX`" are used mainly to store the packing decompression stub. As each section's goals are distinct, the sample's matching rates when considering each one of them is different as well, even if using the same `J-sdhash` tool, as shown in Figure 21. As expected, each implementation choice led to a distinct matching rate. Even distinct packers sections, such as UPX and Themida, resulted in distinct matching rates.

Finding #16: *Each binary section implements a distinct part of the malicious behavior and thus present a distinct similarity score.*

Considering this finding, it is straightforward to hypothesize that separating the distinct sections according to their goals (e.g., the instructions that implement a malicious behavior and the malicious payloads stored as data) might lead to greater similarity metrics. However, there is still a challenge to be considered: Whereas the differences in the matching rates in the distinctly named sections are suggestive, they can only be confirmed by understanding the attributes of each section. As the PE format does not specify any strict rule for section naming, a section named "data" by a malware sample could store in fact instructions. Therefore, to perform a more precise check, we chose to parse the section headers of all PE files and dump their executable and non-executable section bytes in separated files. We then repeated the previously presented clustering experiments with these new files.

Figures 22, 23, 24 and 25 present, respectively, accuracy, precision, recall, and F1-score values obtained in the experiments. All tests were performed considering the `NANO` AV because it was the AV that presented the best results in the aforementioned AV tests. We discovered that this strategy resulted in an overall increase of all considered metrics. We highlight that such improvement is observed in the comparison with AV-assigned labels, which naturally complicates the task, as shown in previous experiments. Therefore, we conclude that such change not only clustered more samples, but actually helped AVs to recognize them as similar.

More specifically, we observe that the behavior of the accuracy metric is very similar to what is observed for the full binary experiment. It starts growing, with some points achieving a 10% higher score than was achieved for the full binary. However, it is still limited by the AVs labels preventing the metric to achieve values higher than 70%. The precision metric results confirm that TP increased when separating data and instructions from the main binary. This also caused the recall metric to be less affected, thus achieving more than 10% higher values at some specific points, especially for higher thresholds, when the samples are expected to be really more similar. These facts resulted in overall slightly better F1 scores.

Finding #17: *Separating instructions from data sections leads to better similarity metrics scores.*

We notice that both the instruction-only as well as the data-only sections outperformed the original baseline, thus showing that mixing instruction and data make similarity detection harder. We believe that in the future when similarity hashing function-based approaches become mainstream, this fact might be purposely exploited by attackers to lower the similarity score of their released malware variants. We consider this is a plausible hypothesis since many attackers already mix code and data to fool disassemblers [15].

Increasing Matching Rates. The previous results have shown that instruction sections are the most discriminant binary components for family clustering procedures using similarity hashing functions. This is reasonable as the instruction sequences present in the binaries are responsible for causing harm and thus characterizing a binary as malicious. Therefore, instructions are a natural focus of any strategy to enhance the classification performance of the functions.

In the forensics context, researchers discovered that common blocks might influence the similarity assessment process of two objects. Many pieces of common data are found to be present in many files of the same and different types. These pieces are related to application-generated content, such as header/footer information, color pallets, font specifications, etc [47]. For instance, Moia et al. [46] show that removing such blocks from the similarity assessment may increase the similarity detection rate in some cases and even filter out many irrelevant matches. Thus, it is plausible to hypothesize that this same reasoning might apply to binary similarity.

We adapted the concept of common blocks to common instructions–i.e., instead of considering a set of bytes representative of a data pattern, we considered them representative of an instruction and its argument. The rationale behind this choice is that by removing common instructions that are present in all (or most) binaries, and thus that do not differentiate them, we allow the similarity hashing function to focus on the file contents that actually make them similar or distinct. In other words, we aim to make similarity scores more representative, since a similarity score considering instructions present in all binaries is naturally lifted due to this similarity. A drawback of the common blocks method is that the common block pattern might be removed from any file region, thus potentially removing more than it was designed to remove (e.g., removing actual data in addition to file headers). This problem is exacerbated when moving to instructions, since the same instruction at distinct file regions might imply distinct high-level behaviors. However, as in the common block's case, we expect the most common instructions to be statistically more significant in

Table 6: **Binary Sections of the malware samples in our dataset.** Each sample presents its own set of executable binary sections.

Section	Prevalence	Section	Prevalence	Section	Prevalence	Section	Prevalence
data	97%	.bss	97%	.idata	92%	.reloc	90%
bss	85%	..tls	73%	itext	72%	.rdata	58%
.rsrc	51%	UPX1	34%	UPX0	34%	.data	34%
.code	29%	.edata	20%	.text	18%	Others	17%

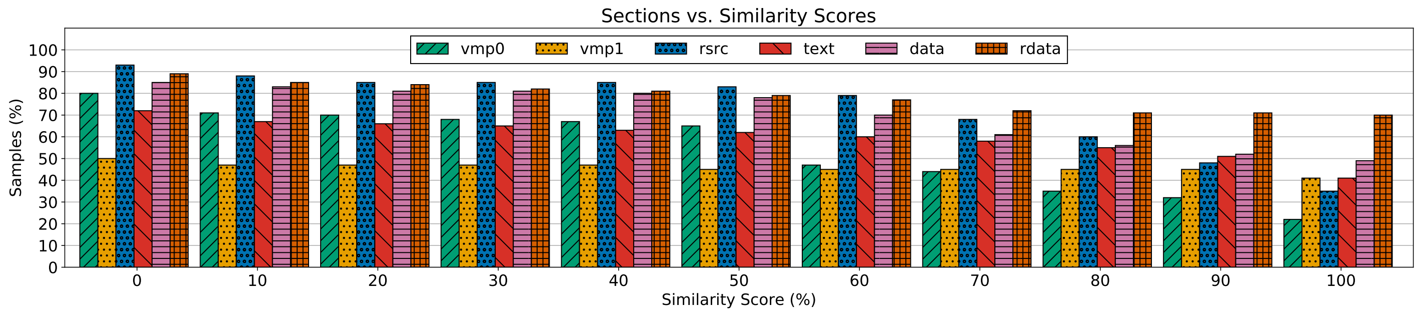

Figure 21: **Similarity Matches per Section.** Each section presents a distinct rate.

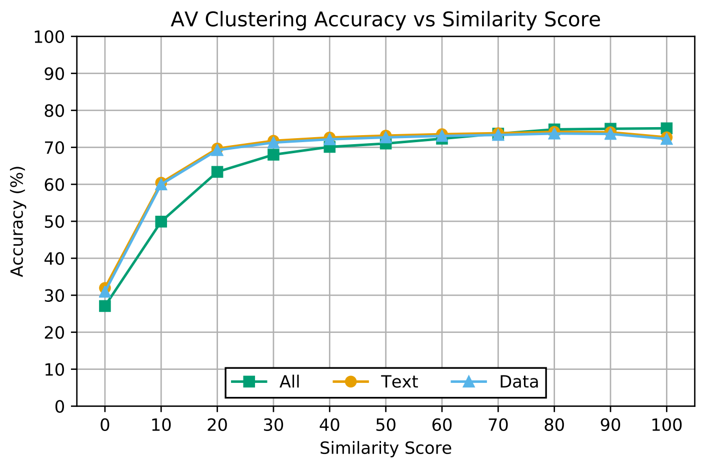

Figure 22: **Binary Sections Accuracy**

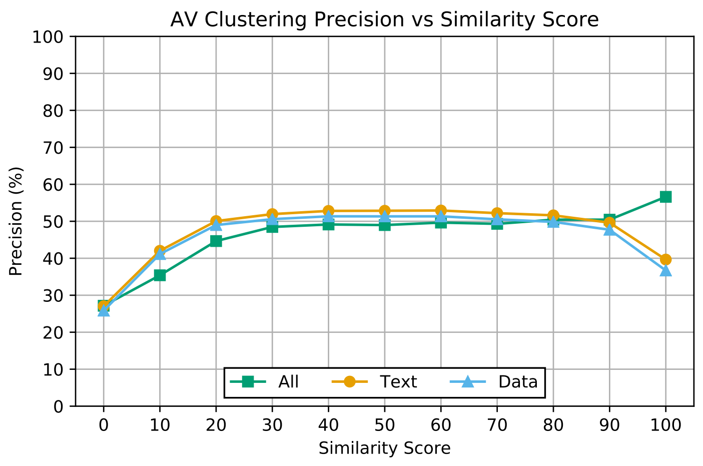

Figure 23: **Binary Sections Precision**

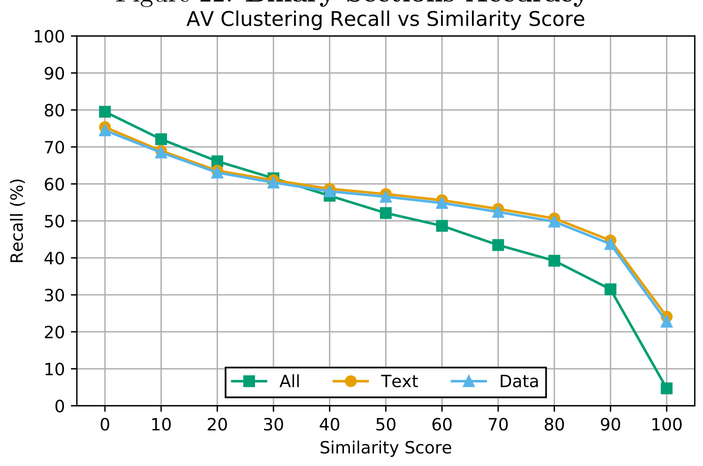

Figure 24: **Binary Sections Recall**

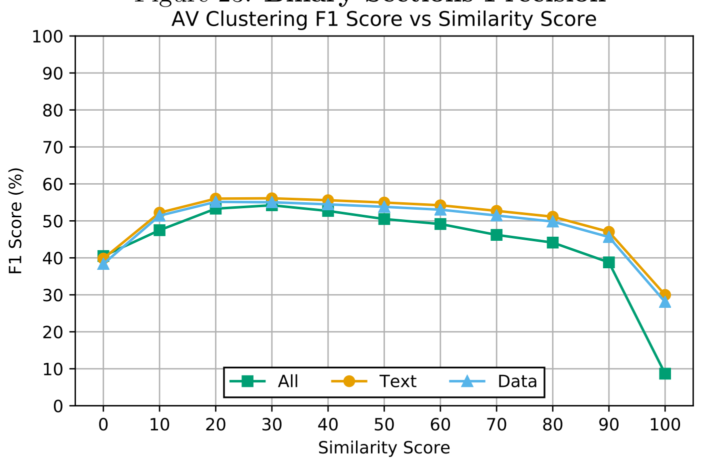

Figure 25: **Binary Sections F1 Score**

non-essential code regions, such that we end up observing metrics improvements.

Considering the proposed method adaptation, we repeated the presented experiments now considering each binary instruction as a possible common block. In total, we considered 377 million unique combinations of instruction opcodes and arguments present in the disassembly of the executable sections of the malware binaries. Table 7 exemplifies the top 10 instructions most found in our datasets and that might be removed to improve malware detection rates.

Table 7: **Common Blocks.** Top 10 most removed instructions.

Instruction	Prevalence	Instruction	Prevalence
`push <reg>`	95.34%	`nop`	89.84%
`pop <reg>`	94.77%	`int3`	89.24%
`inc <reg>`	94.36%	`cltd`	88.43%
`dec <reg>`	94.19%	leave	83.49%
`ret`	94.18%	`in`	83.38%

We notice that the most prevalent instructions, individually or in combination, really reflect common binary constructions. More specifically, we notice their relation with function prologues (`push`) and epilogues (`pop+ret`). By removing these binary function's surrounding instructions, we allow the similarity hashing function to focus on the similarity of the actual binary function's content.

To evaluate our hypothesis in practice, we varied the ratio of the removed instructions (in % of the most common instructions) to identify the binary portion that most discriminates the binaries. Figures 26, 27, 28, and 29 presents, respectively, accuracy, precision,recall, and F1-score values obtained in the experiment while keeping 70%, 80%, 90%, and 100% of the binary instructions (thus removing from 0% to 30% of the most common instructions).

This experiment sheds light on two important aspects of similarity hashing functions. First, we notice that, whereas removing a distinct number of common instructions did not improve the overall metrics, it also did not decrease them. This shows that approximate matching might also be successful in identifying similarity of corrupted data or code excerpts capture from partial memory dumps, which is key for analyzing malware. Further, we notice that the 70% threshold (removing the 30% most common instructions and keeping the remaining 70%) leads to an overall performance increase. We again highlight that this result is observed even considering the limits imposed by the AV labeling procedures, which shows that this strategy not only clustered more samples but also helped similarity hashing functions and the AV to agree on their clusters.

The behavior of all metrics in this experiment was very similar to in the previous one, but now achieving only slightly better results, which is expected, since the result has already been significantly improved by the adoption of the previous techniques (e.g., separating instructions from other sections). Even though, precision was the metric that most increased, which demonstrates that this technique has the potential to still increase the detection of similar samples in agreement with AV labels.

The obtained result is explained by the fact that by feeding the similarity hash with less data (70% of the total), the bytes are distributed distinctly within the similarity hashing function's clustering buckets (see Section 2). This new bucket organization leads to distinct digest values, that lead to distinct similarity scores, and thus to distinct clusters.

Finding #18: *Removing common instructions forces similarity hashing function's internal buckets reorganization, which leads to similarity metrics increase.*

5 Guidelines for the Application of Similarity Hashing Functions

With this work's development, we discovered that although similarity hashing functions are already popular solutions in many contexts, there are still no guidelines for their application, with distinct researchers and solutions taking ad-hoc decisions on parameter selection. Our experiments presented results provide interesting insights about the scenarios in which distinct parameters and approaches lead to better usage of such functions. Therefore, aiming to help to bridge this guidelines gap, we propose some rules to be followed when designing and performing experiments with similarity hashing functions.

1. Be aware of the existing trade-offs when **choosing a similarity hashing function**.
   (a) **J-sdhash clusters more samples than ssdeep**, thus being more suitable for tasks that require greater sample's coverage.
   (b) **Ssdeep is faster than J-sdhash** while presenting slightly lower similarity scores, thus being more suitable for performance-constrained environments.
      i. **The cost of loading a database for matching is non-negligible** for any of the two functions.
2. Always explicitly **choose a threshold value** and do not simply rely on tool's default parameters and heterogeneous meanings. A threshold must reflect what and how much the researchers envision as similar or not.
   (a) Be aware that **distinct threshold values might lead to completely distinct dataset characterizations**, thus making you to take opposite conclusions with its variations.
   (b) Distinct threshold values perform better for distinct tasks, with **smaller threshold values performing better for malware triaging** and **greater threshold values performing better for malware remediation and response**.
3. **Do not mix malware detection and malware family clustering**, since they are distinct tasks, with distinct drawbacks.

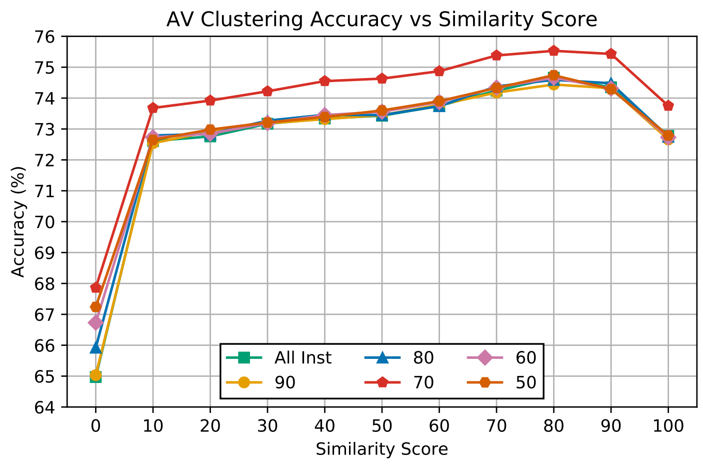

Figure 26: **Accuracy Removing Blocks**

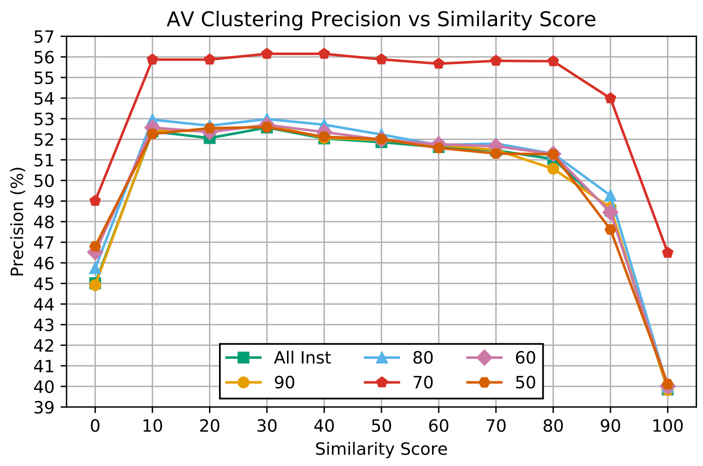

Figure 27: **Precision Removing Blocks**

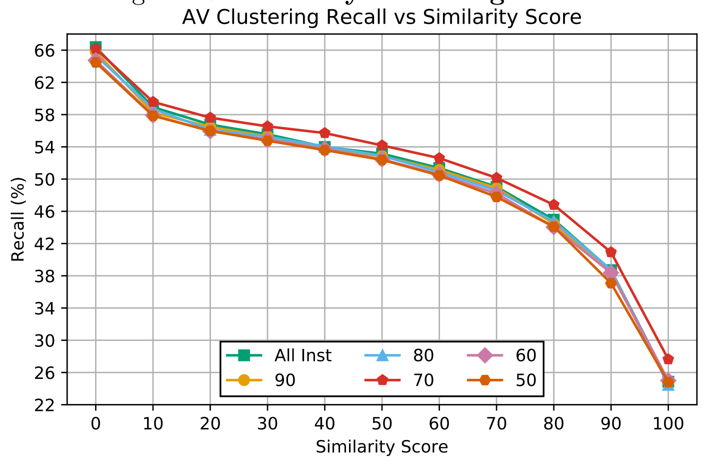

Figure 28: **Recall Removing Blocks**

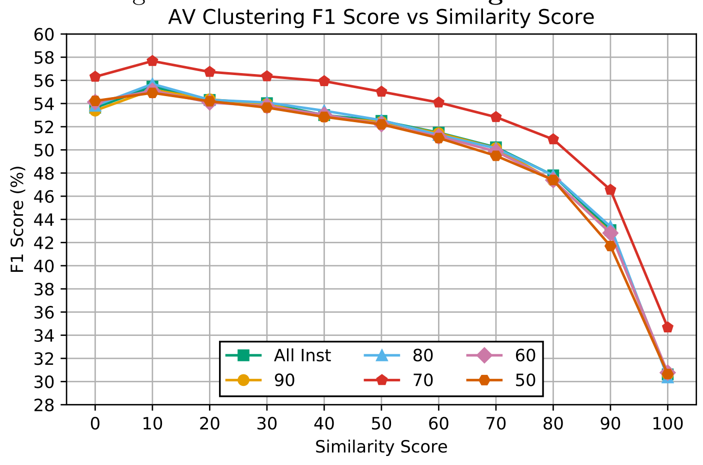

Figure 29: **F1 Removing Blocks**

4. When **detecting malware samples**:
    (a) **Considering sample's similarity increases AV's detection**, since samples not detected by the AV can now be detected by their similarity with previously detected samples.
    (b) **Intermediate threshold values** (e.g., 50%) **present the best trade-off** between malware detection and false positives.
    (c) **A good threshold choice** (e.g., 50%) **allows a similarity-based AV to detect a whole dataset without adding all of them to a database**.

5. When **classifying malware samples in families**:
    (a) Notice that **A committee of AVs is a better ground-truth than individual AVs**, since it reduces the heterogeneity of the AV's labels.
    (b) Individual AVs should be avoided because **distinct AVs lead to distinct results**, since **they rely on distinct classification criteria**.
        i. Many **AVs classify samples as similar based on their file type and/or packer**, regardless their actual content.
        ii. **Avoid directly comparing packed and unpacked samples** and prefer unpacking the samples whenever possible to maximize similarity scores.
    (c) The application of **individual AVs lead to low intra-cluster agreement**, since the clusters created by the similarity hashing functions will be distinct from the clusters generated by the AVs.
    (d) If **using individual AVs are still the only possibility**:
        i. Be aware that **the AV that most detects samples is not necessarily the one that most clusters them**, thus additional selection criteria are required.
        ii. Choose the AVs that **best-fits the similarity hashing function results**, so it allows one to check function's improvements without the uncertainty of AV results.
    (e) Remind that you can **optimize the use of hash functions** to achieve better results.
        i. **Distinct binary sections present distinct similarity rates**, as they implement distinct malicious behaviors and/or components.
        ii. **Separating code and data increases similarity scores**, as it avoid the distinct patterns exhibited in these two kinds of sections to be mixed.
            A. Be careful when identifying executable sections as **packed code might turn sections executable only in runtime.**
        iii. **Removing common instruction increases similarity metrics**, as it supplies the function with less data and it cause internal buckets rearrangements.

6 Discussion

In this section, we discuss the limits of our evaluations and the implications of our findings.
The Importance of the Threshold Value. Our experiments have shown that the threshold value affects the number of clustered samples and thus the conclusions when characterizing a dataset. Therefore, we highlight the importance of researchers reporting the used threshold in their experiments to allow reproducibility and fair comparisons. There is currently no guideline for the threshold definition. We consider that this definition is a key open question towards the adoption of similarity metrics as standard for malware experiments and solutions. We expect our findings might help in bridging this gap and incentivizing future guidelines development. Meanwhile, we suggest researchers clearly stating their considered threshold values so as to allow reproducibility. Moreover, we suggest them open-sourcing their similarity solutions since the threshold value meaning changes from solution to solution.
Defining Ground-Truth is Hard. Our results have shown that the clusters originated from AVs and from similarity hashing functions differ significantly. We discovered that the main reason is that AVs generate labels based on detection heuristics and not on binary similarity. Therefore, there is a significant opportunity for the development of new methods and strategies for ground-truth definition. We consider it as an important step for allowing malware similarity experiments to be validated and new clustering/identification approaches to be compared to each other on the same basis.
The Advantages of Alternative Malware Representations. Due to the nature of the similarity hashing functions, our proposed AV only needs to upload the digest of a suspicious file to be scanned instead of the entire file content, which results in privacy and performance gains. This is made possible because the uploaded digest is, in fact, a short representation of the scanned file, which saves network bandwidth and presents itself as an effective and scalable alternative representation for malware detection.
Similarity hashing for triaging malware. A straightforward application of similarity hashing in the context of malware analysis and detection is to triage samples, i.e., to verify the sample is known or not. This task is often performed using cryptographic hashes (e.g., MD5 or SHA), which limits its application mostly to static databases. If similarity hashes were used, the triage procedure could be turned into the verification of whether there is a known way to handle that sample, despite its being known or unknown. Similarity hashes could then be used, for instance, to cluster new samples to the existing ones and thus the same procedures applied to that cluster would be applied to the new sample. This would allow the scaling of multiple triaging ap-

proaches, such as Google SafeBrowsing [29], as users would not need to upload entire files (but only their similarity digests) to the server to have initial feedback about the file security.

Scaling Malware Variants Identification. Multiple approaches have been proposed over time to identify malware variants. Some of them, such as those relying on dynamic analysis [14], are computationally expensive, despite effective, thus being not practical to handle large-scale databases. The reliance on similarity hashing functions proposed in this work allows an efficient and yet effective solution to handle large-scale databases. In our approach, suspicious files are matched only against representative samples of each cluster. Therefore, costly approaches can be employed at the server-side to define the cluster's "leader" whereas our approach can be used to fast calculate the similarity between these leaders and the suspicious file.

identifying executable sections might be challenging. The major drawback of relying on individual sections is that originally read-only sections can be turned into an executable one in runtime by malware samples. This type of information is only available for dynamic analysis procedures and not for static approaches like the ones proposed in this paper. Therefore, the best way of benefiting from this type of approach is to not perform this task statically at the client-side, but to outsource it to a more powerful agent, such as a cloud-based AV server, which might execute and trace the sample before matching its executable sections.

Limitations Whereas we believe our work covered a significant number of challenges in the application of similarity hashing functions to malware decision processes, we are aware that it has some limitations. For instance, ssdeep and J-sdhash are not the only similarity hashing functions that could be selected. We considered them in our evaluation due to their popularity, but other functions, such as TLSH variations, should also be evaluated to draw a broader landscape of the malware classification subject. Moreover, our dataset is only a limited view of the larger problem of malware handling. We do not claim that our findings generalize from this dataset to any other. Instead, our claim is that this dataset particularly highlights the challenges of malware classification using similarity hashing functions. We demonstrated that the challenges we pointed really occurred in a real dataset and thus they should be considered by professionals and researchers as they might appear during their investigative activities. However, other datasets might reveal distinct challenges, so more research is warranted.

Future Work. This work has shown alternative approaches for the application of similarity hashing functions. We believe that our findings might help to broaden the usage scenarios of this type of function. However, this work is not exhaustive and more research is warranted to investigate additional issues. For instance, whereas our work encompassed only MS Windows binaries, the same experiments should be repeated to assess whether the same conclusions hold for Linux and Android applications. In future work, we plan to investigate the impact of considering common blocks of multiple instructions in addition to single instructions and new sources of labels ground-truths.

7 Related Work

In this section, we present related work to better position our contributions. Whereas the main usage of similarity hashing functions is for sample triaging and clustering [25, 30], we below focus more specialized developments of similarity hashing.

Approximate Matching. Many functions were developed over the years trying to provide a lightweight solution, having as short as possible digests with generation and comparison times just as fast as traditional hash functions, such as SHA-1, SHA-2, etc. The more prominent functions include ssdeep [36], sdhash [60], mrsh-v2 [17], TLSH [54], and the most recent one, LZJD [59]. However, the most popular and the chosen ones for this work are ssdeep and sdhash, for being the first functions of their kind and target of constant research, regarding improvements [16, 46, 45] and discussing their detection capabilities [5, 20, 19].

Approximate Matching and Malware. The application of similarity hashing functions to binary files was well studied by Pagani et al [56], which investigates use cases such as library identification and binary recompilation. Malware are particular types of binaries whose analysis tasks can benefit from the application of similarity hash functions. These have already been reported to be applied in forensic procedures [64] and for hunting ransomware [50]. Their major application field, however, is the identification of malware variants [4].

Drawbacks of Approximate Matching. The application of similarity hashing functions presents some drawbacks, such as the ones mentioned by Pagani et al [56], related to a change in the binary structure of files. Since the functions employed in this work are meant to find similarity in the bytewise level, changes in the compiler during binary creation or due to packing and compressing techniques will negatively impact the use of the similarity hashing function in the assessment of similarity. A drawback related to their application in practice is the resulting cluster diversity. The application of similarity hash functions to IoT binaries produced the same trade-offs between cluster sizes and performance presented in this work [6].

Design of new similarity hash functions. A way to advance malware similarity identification solutions is to develop new similarity hash functions. Previous research works have identified the requirements for a good similarity hash function [55] and how it can inspire new applications [35], but there are still few works targeting the similarity of executable binaries] [39], and more specifically proposing new functions that are malware-targeted [38]. We expect that this study might foster new research and that our findings might support their developments.

Malware Variants Identification. The identification of malware variants is a challenging task and multiple approaches have been proposed to address it: the use of n-grams [26], visual analytics [57], call graphs (CGs) [9], graph embedding [72], and dynamic analysis [2]. All of these approaches, however, are more computationally expensive than the here proposed application of hash functions. Additionally, nothing prevents these approaches to be used as a complement to ours if applied over the clusters identified by the similarity hash functions.

Other Matching Approaches. Malware variants identification is not the only method that works by matching similar patterns. The same task is performed in the context of basic blocks comparison [1] and code reuse identification [69]. Similarity hashing is an important step part of these approaches and our findings are also valid to be applied along with them.

Matching Strategies. A key contribution of this work is to discuss matching criteria and strategies to effectively detecting common binaries. Whereas our focus is on malware detection, previous work presented related discussion in other fields, such as in Windows modules identification [41]. The application of this kind of strategy to malware samples is an open problem.

Other Labeling Strategies. In addition to AVs, other solutions might provide ground-truth information about malware families. For instance, many machine-based approaches have been proposed recently to address the problem of weak malware labels [73]. These approaches are varied, and might rely from static strings extracted from the samples [67], to artifacts extracted during dynamic analysis procedures [44]. Other approaches try to overcome malware classification challenges via the observation of their behaviors [49]. A major drawback of these approaches is that they are more costly (e.g., requiring runtime analysis) than relying on AV labels. Also, the implementation of these approaches is not often available during the practice of an analyst and/or forensic expert. Thus, we focused this work on the use of the widely-available AV labels.

Cloud AVs. A key insight of our evaluation is that the similarity hash-based matching will be outsourced to a cloud-based AV. The idea of a cloud AV is not new, but most solutions only move the same technology to the cloud and do not change how detection is performed [24]. In some cases, the detection is even outsourced to third part AVs [33]. In our proposal, in addition to the processing tasks, we also change the file scanning representation from signatures to similarity hashes and scores. This allows us to benefit from other cloud-based strategies, such as distributed data collection [23].

The bias towards the easy cases. In this paper, we tackled the problem of identifying similar malware samples in a real scenario, which brings many difficulties and limits the obtained results. Our goal with this evaluation is to highlight how challenging a theoretically-feasible task might become in a real scenario. In this sense, our evaluation follows the philosophy adopted in previous work to demonstrated that evaluating solution only using the "easy-to-classify" samples is not enough for a complete understanding of the problem. Previous works have already demonstrated that the high accuracy rates reported for both traditional [37] as well as emerging [7] (textures) malware clustering approaches might not be reproducible depending on how challenging the evaluated dataset is.

8 Conclusion

In this work, we investigated the limits and benefits of applying similarity hashing to clustering routines to enhance malware detection procedures applied to real-world datasets. We proposed an ideal model of a similarity hashing-based AntiVirus (AV) that applied two distinct similarity hash functions to a dataset of real malware samples collected over four years. We discovered that considering similarity scores might improve standard AV's detection rates by up to 40% without raising False Positives. The centralized database model adopted by our AV along with our findings suggests that approaches so-far consider impractical, such as complex clustering procedures, might now be successfully implemented at the AV server level. On the other hand, we discovered that leveraging AV's labels as ground-truth for malware similarity identification is challenging and requires additional steps to achieve reasonable results. For instance, we showed that considering only the most-representative binary instructions, i.e., the unique instructions of each binary or the ones with fewer occurrences in different files), leads to better similarity metrics than considering the entire binary, thus a better fit between the similarity hashing function and the considered AV.

Acknowledgements. This study was financed in part by the Brazilian National Counsel of Technological and Scientific Development (CNPq - process n. 164745/2017-3), the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES – Finance Code 001) and the Santander International Mobility Program for Graduate Students.

Reproducibility. All developed source codes for this research are available at: `https://github.com/marcusbotacin/Binary.Similarity` The list of considered malware samples is available at `https://github.com/marcusbotacin/malware-data` Goodware information is available at `https://github.com/marcusbotacin/Application.Installers.Overview`

References

[1] F. Adkins, L. Jones, M. Carlisle, and J. Upchurch. Heuristic malware detection via basic block comparison. In *2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*, pages 11–18, 2013.

[2] E. M. S. Alkhateeb. Dynamic malware detection using api similarity. In *2017 IEEE International Conference on Computer and Information Technology (CIT)*, pages 297–301, 2017.

[3] Flora Amato, Aniello Castiglione, Giovanni Cozzolino, and Fabio Narducci. A semantic-based methodology for digital forensics analysis. *Journal of Parallel and Distributed Computing*, 138:172–177, 2020.

[4] A. Azab, R. Layton, M. Alazab, and J. Oliver. Mining malware to detect variants. In *2014 Fifth Cybercrime and Trustworthy Computing Conference*, pages 44–53, 2014.

[5] Harald Baier and Frank Breitinger. Security aspects of piecewise hashing in computer forensics. In *IT Security Incident Management and IT Forensics (IMF), 2011 Sixth International Conference on*, pages 21–36. IEEE, 2011.

[6] Marton Bak, Dorottya Papp1, Csongor Tamas, and Levente Buttyan. Clustering iot malware based on binary similarity. `https://www.crysys.hu/publications/files/setit/cpaper_bme_BakPTB20dissect.pdf`, 2020.

[7] Tamy Beppler, Marcus Botacin, Fabrício J. O. Ceschin, Luiz E. S. Oliveira, and André Grégio. L(a)ying in (test)bed. In Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis, editors, *Information Security*, pages 381–401, Cham, 2019. Springer International Publishing.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[9] K. Blokhin, J. Saxe, and D. Mentis. Malware similarity identification using call graph based system call subsequence features. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 6–10, 2013.

[10] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[11] Marcus Botacin, Hojjat Aghakhani, Stefano Ortolani, Christopher Kruegel, Giovanni Vigna, Daniela Oliveira, Paulo Lício De Geus, and André Grégio. One size does not fit all: A longitudinal analysis of brazilian financial malware. *ACM Trans. Priv. Secur.*, 24(2), January 2021.

[12] Marcus Botacin, Giovanni Bertão, Paulo de Geus, André Grégio, Christopher Kruegel, and Giovanni Vigna. On the security of application installers and online software repositories. In Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 192–214, Cham, 2020. Springer International Publishing.

[13] Marcus Botacin, Fabricio Ceschin, Paulo [de Geus], and André Grégio. We need to talk about antiviruses: challenges & pitfalls of av evaluations. *Computers & Security*, 95:101859, 2020.

[14] Marcus Botacin, Paulo de Geus, and André Grégio. Malware variants identification in practice. `https://sbseg2019.ime.usp.br/anais/195666.pdf`, 2019.

[15] Marcus Botacin, Lucas Galante, Paulo de Geus, and André Grégio. Revenge is a dish served cold: Debug-oriented malware decompilation and reassembly. In *Proceedings of the 3rd Reversing and Offensive-Oriented Trends Symposium*, ROOTS'19, New York, NY, USA, 2019. Association for Computing Machinery.

[16] Frank Breitinger and Harald Baier. *Performance Issues About Context-Triggered Piecewise Hashing*, pages 141–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[17] Frank Breitinger and Harald Baier. *Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2*, pages 167–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[18] Frank Breitinger, Barbara Guttman, Michael McCarrin, Vassil Roussev, and Douglas White. Approximate matching: definition and terminology. *NIST Special Publication*, 800:168, 2014.

[19] Frank Breitinger and Vassil Roussev. Automated evaluation of approximate matching algorithms on real data. *Digital Investigation*, 11:S10–S17, 2014.

[20] Frank Breitinger, Georgios Stivaktakis, and Vassil Roussev. Evaluating detection error trade-offs for bytewise approximate matching algorithms. *Digital Investigation*, 11(2):81–89, 2014.

[21] F. Ceschin, F. Pinage, M. Castilho, D. Menotti, L. S. Oliveira, and A. Gregio. The need for speed: An analysis of brazilian malware classifers. *IEEE Security & Privacy*, 16(6):31–41, Nov.-Dec. 2018.

[22] CNN. Nearly 1 million new malware threats released every day. `https://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/`, 2015.

[23] M. Dev, H. Gupta, S. Mehta, and B. Balamurugan. Cache implementation using collective intelligence on cloud based antivirus architecture. In *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pages 593–595, 2016.

[24] Dimitris Deyannis, Eva Papadogiannaki, Giorgos Kalivianakis, Giorgos Vasiliadis, and Sotiris Ioannidis. Trustav: Practical and privacy preserving malware analysis in the cloud. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, CODASPY '20, page 39–48, New York, NY, USA, 2020. Association for Computing Machinery.

[25] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla†. discovre: Efficient cross-architecture identification of bugs in binary code - ndss 17, 2017.

[26] Z. Fuyong and Z. Tiezhu. Malware detection and classification based on n-grams attribute similarity. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, volume 1, pages 793–796, 2017.

[27] Lucas Galante, Marcus Botacin, André Grégio, and Paulo de Geus. Malicious linux binaries: A landscape. `https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4160`, 2018.

[28] Google. How google play protect kept users safe in 2019. `https://security.googleblog.com/2020/03/how-google-play-protect-kept-users-safe.html`, 2019.

[29] Google. Improved malware protection for users in the advanced protection program. `https://security.googleblog.com/2020/09/improved-malware-protection-for-users.html`, 2020.

[30] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, page 1057–1072, USA, 2015. USENIX Association.

[31] Vikram S Harichandran, Frank Breitinger, and Ibrahim Baggili. Bytewise approximate matching: The good, the bad, and the unknown. *The Journal of Digital Forensics, Security and Law: JDFSL*, 11(2):59, 2016.

[32] Qing He, Hai Xia Gu, Qin Wei, and Xu Wang. A novel dbscan based on binary local sensitive hashing and binary-knn representation. *Advances in Multimedia*, 2017:3695323, Dec 2017.

[33] Chris Jarabek, David Barrera, and John Aycock. Thinav: Truly lightweight mobile cloud-based anti-malware. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, page 209–218, New York, NY, USA, 2012. Association for Computing Machinery.

[34] André Kameyama, Vitor Moia, and Marco Aurélio Amaral Henriques. Aperfeiçoamento da ferramenta sdhash para identificação de artefatos similares em investigações forenses. `https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4161`, 2018.

[35] ElMouatez Billah Karbab, Mourad Debbabi, and Djedjiga Mouheb. Fingerprinting android packaging: Generating dnas for malware detection. *Digital Investigation*, 18:S33–S45, 2016.

[36] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3:91–97, 2006.

[37] Peng Li, Limin Liu, Debin Gao, and Michael K. Reiter. On challenges in evaluating malware clustering. In Somesh Jha, Robin Sommer, and Christian Kreibich, editors, *Recent Advances in Intrusion Detection*, pages 238–255, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[38] Yuping Li, Sathya Chandran Sundaramurthy, Alexandru G. Bardas, Xinming Ou, Doina Caragea, Xin Hu, and Jiyong Jang. Experimental study of fuzzy hashing in malware clustering analysis. In *Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test*, CSET'15, page 8, USA, 2015. USENIX Association.

[39] Lorenz Liebler and Harald Baier. Towards exact and inexact approximate matching of executable binaries. *Digital Investigation*, 28:S12–S21, 2019.

[40] Jacques Linden, Raymond Marquis, Silvia Bozza, and Franco Taroni. Dynamic signatures: A review of dynamic feature variation and forensic methodology. *Forensic Science International*, 291:216–229, 2018.

[41] Miguel Martín-Pérez, Ricardo J. Rodríguez, and Davide Balzarotti. Pre-processing memory dumps to improve similarity score of windows modules. *Computers & Security*, 101:102119, 2021.

[42] Miguel Martín-Pérez, Ricardo J. Rodríguez, and Frank Breitinger. Bringing order to approximate matching: Classification and attacks on similarity digest algorithms. *Forensic Science International: Digital Investigation*, 36:301120, 2021. DFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth Annual DFRWS Europe Conference.

[43] Fernando Mercês. Grouping linux iot malware samples with trend micro elf hash. `https://blog.trendmicro.com/trendlabs-security-intelligence/grouping-linux-iot-malware-samples-with-trend-micro-elf-hash/`, 2020.

[44] Aziz Mohaisen and Omar Alrawi. Amal: High-fidelity, behavior-based automated malware analysis and classification. In Kyung-Hyune Rhee and Jeong Hyun Yi, editors, *Information Security Applications*, pages 107–121, Cham, 2015. Springer International Publishing.

[45] Vitor Hugo Galhardo Moia. *A study on approximate matching for similarity search: techniques, limitations and improvements for digital forensic investigations*. PhD dissertation, School of Electrical and Computer Engineering - University of Campinas, Campinas, SP, Brazil, 2020.

[46] Vitor Hugo Galhardo Moia, Frank Breitinger, and Marco Aurélio Amaral Henriques. Understanding the effects of removing common blocks on approximate matching scores under different scenarios for digital forensic investigations. In *XIX Brazilian Symposium on information and computational systems security*, pages 1–14. Brazilian Computer Society (SBC) SÃ£ o Paulo-SP, Brazil, 2019.

[47] Vitor Hugo Galhardo Moia, Frank Breitinger, and Marco Aurélio Amaral Henriques. The impact of excluding common blocks for approximate matching. *Computers & Security*, 89:101676, 2020.

[48] Vitor Hugo Galhardo Moia and Marco Aurélio Amaral Henriques. Similarity digest search: A survey and comparative analysis of strategies to perform known file filtering using approximate matching. *Security and Communication Networks*, 2017, 2017.

[49] Azqa Nadeem, Christian Hammerschmidt, Carlos H. Gañán, and Sicco Verwer. *Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families*, pages 381–409. Springer International Publishing, Cham, 2021.

[50] N. Naik, P. Jenkins, N. Savage, and L. Yang. Cyberthreat hunting - part 1: Triaging ransomware using fuzzy hashing, import hashing and yara rules. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2019.

[51] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, and J. Song. Augmented yara rules fused with fuzzy hashing in ransomware triaging. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 625–632, 2019.

[52] Landon Curt Noll. Fowler/noll/vo (fnv) hash. `http://www.isthe.com/chongo/tech/comp/fnv/index.html`, 2012. Accessed 2020 Apr 14.

[53] Markus Oberhumer, Laszlo Molnar, and John Reiser. the ultimate packer for executables. `https://upx.github.io/`, 2017.

[54] Jonathan Oliver, Chun Cheng, and Yanggui Chen. TLSH–a locality sensitive hash. In *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*, pages 7–13. IEEE, 2013.

[55] Jonathan Oliver and Josiah Hagen. On designing the elements of a fuzzy hashing scheme. `https://www.malwareconference.org/index.php/en/2019-malware-conference-proceedings/2019-malware-conference/session-2-case-studies-analysis-and-industry-view/05-4720-pdf/detail`, 2019.

[56] Fabio Pagani, Matteo Dell'Amico, and Davide Balzarotti. Beyond precision and recall: Understanding uses (and misuses) of similarity hashes in binary analysis. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, page 354–365, New York, NY, USA, 2018. Association for Computing Machinery.

[57] A. Paturi, M. Cherukuri, J. Donahue, and S. Mukkamala. Mobile malware visual analytics and similarities of attack toolkits (malware gene analysis). In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 149–154, 2013.

[58] Matt Pietrek. An in-depth look into the win32 portable executable file format. `https://docs.microsoft.com/en-us/archive/msdn-magazine/2002/february/inside-windows-win32-portable-executable-file-format-in-detail`, 2002.

[59] Edward Raff and Charles Nicholas. Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*, 24:34–49, 2018.

[60] Vassil Roussev. Data fingerprinting with similarity digests. In *IFIP International Conf. on Digital Forensics*, pages 207–226. Springer, 2010.

[61] Vassil Roussev. An evaluation of forensic similarity hashes. *Digital investigation*, 8:34–41, 2011.

[62] Vassil Roussev. An evaluation of forensic similarity hashes. *Digital Investigation*, 8:S34 – S41, 2011. The Proceedings of the Eleventh Annual DFRWS Conference.

[63] Vassil Roussev and Candice Quates. sdhash tutorial: Release 0.8. `http://roussev.net/sdhash/tutorial/sdhash-tutorial.pdf`, 2013. Accessed 2016 Set 13.

[64] N. Sarantinos, C. Benzaïd, O. Arabiat, and A. Al-Nemrat. Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1782–1787, 2016.

[65] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 230–253, Cham, 2016. Springer International Publishing.

[66] Ian Shiel and Stephen O'Shaughnessy. Improving file-level fuzzy hashes for malware variant classification. *Digital Investigation*, 28:S88–S94, 2019.

[67] Prasha Shrestha, Suraj Maharjan, Gabriela Ramírez de la Rosa, Alan Sprague, Thamar Solorio, and Gary Warner. Using string information for malware family identification. In Ana L.C. Bazzan and Karim Pichara, editors, *Advances in Artificial Intelligence – IBERAMIA 2014*, pages 686–697, Cham, 2014. Springer International Publishing.

[68] Esko Ukkonen. On approximate string matching. In *International Conference on Fundamentals of Computation Theory*, pages 487–495. Springer, 1983.

[69] J. Upchurch and X. Zhou. Malware provenance: code reuse detection in malicious software at scale. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 1–9, 2016.

[70] VirusTotal. Virustotal. `https://www.virustotal.com`, 2018.

[71] Y. Wu, J. Guo, and X. Zhang. A linear dbscan algorithm based on lsh. In *2007 International Conference on Machine Learning and Cybernetics*, volume 5, pages 2608–2614, 2007.

[72] Xiaochuan Zhang, Wenjie Sun, Jianmin Pang, Fudong Liu, and Zhen Ma. Similarity metric method for binary basic blocks of cross-instruction set architecture. `https://archive.bar/pdfs/bar2020-preprint2.pdf`, 2020.

[73] Y. Zhang, Y. Sui, S. Pan, Z. Zheng, B. Ning, I. Tsang, and W. Zhou. Familial clustering for weakly-labeled android malware using hybrid representation learning. *IEEE Transactions on Information Forensics and Security*, 15:3401–3414, 2020.