Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Improving file-level fuzzy hashes for malware variant classification

Ian Shiel, Stephen O'Shaughnessy*

TU Dublin, Blanchardstown, Dublin, Ireland

A R T I C L E I N F O

Article history:

Keywords:
Malware
Classification
Similarity digest
Fuzzy hash
File-level
Section-level
PE file

A B S T R A C T

Malware analysts need to be able to accurately and swiftly predict family membership as well as to determine that a suspect file contains malicious content. Previous research has shown that fuzzy hashing can be used to determine whether a file is malicious and to cluster like files together, but it does not specifically address the problem of malware variant classification. Existing tools such as VirusTotal maintain file and section level cryptographic hashes and ssdeep file digests but they do not maintain section-level similarity hashes or provide a means to submit similarity hashes and compare them to previously analyzed samples.

This paper presents a study on the feasibility of using section-level similarity hashing as a means of classifying malware variants. The aim of the study was to produce a method to overcome the limitations of file-level similarity hashing, such as poor performance against obfuscated malware. Section-level similarity hashing involves splitting malware executables into their binary headers and sections and applying a similarity digest on each resulting binary chunk. Experiments with known malware families were conducted using file and section level digests where each method was used to predict malware family membership. The performance of both methods was evaluated using precision, recall and accuracy metrics.

The results show that similarity digests can be used to classify malware in Windows Portable Executable (PE) files and that section-level hashing and comparison produces considerably better results than at file-level.

Introduction

Malware variants

A high percentage of current malware is not “new” per se but variants of existing malware families. According to the 2017 Symantec Internet Security Threat Report (Symantec ISTR, 2018, 2018) published in March 2018, the total number of malware variants detected increased by 88% from just over 357M in 2016 to almost 670M in 2017. Variants of the “Kotver” Trojan accounted for 78% of this increase.

Malware analysts need to be able to quickly and accurately predict malware family membership as well as to determine that an executable contains malicious content. During static analysis, an analyst may compare a cryptographic hash of the suspect file to a set of known hashes to determine whether the file has previously been identified. Cryptographic hashes such as MD5 and SHA1 have long been used in digital forensics to verify that two objects are identical and to prove integrity (e.g. a forensic investigation has not modified the original evidence).

Malware variants defeat traditional cryptographic hash signatures as a change in a single bit of the file produces a very different hash result. The continuing increase in malware variants can largely be attributed to the ease of access to automated obfuscation tools. Malware attackers no longer need to have an in-depth knowledge of file or code modification, as these tools essentially do all of this through a graphical user interface. The result is an alarming level of modified malware variants appearing on a daily basis, which has vastly increased the number of suspect files that must be manually examined, the complexity of the triage effort and therefore the time and cost.

Similarity digests have potential use in the classification of malware variants, but there are limitations to be overcome.

Similarity digests

While malware variants will not be identical to each other, they

* Corresponding author.
E-mail address: stephen.o'shaughnessy@itb.ie (S. O'Shaughnessy).

https://doi.org/10.1016/j.diin.2019.01.018

should be similar, for example, large parts of the code may be the same as other variants. Cryptographic hashes are not designed to identify similarities but there are other hashing mechanisms which perform this function. These mechanisms are known as “similarity digests” or “fuzzy hashes” and they can be used not only to identify similarities but also to provide a measure of how similar files and objects are to each other. There are a number of fuzzy hashing algorithms, two of the best known are Kornblum's' “ssdeep” (Kornblum, 2006) and Roussev's “sdhash” (Roussev, 2010). These hashes output a score between 0 and 100 to indicate similarity between the compared objects. 0 represents no or very little similarity and 100 represents very high similarity or identicality. A given input will always produce the same similarity hash, but this does not mean that inputs that produce the same similarity hash are identical – just that they are very similar. Identicality is checked with a cryptographic hash and this should be done before using a similarity hash as it is much faster to compute and comparison involves a single keyed-lookup which is not the case with a similarity hash as many inputs can produce the same hash.

While cryptographic hashes have been used in forensics and malware analysis for identity, integrity and classification purposes, fuzzy hashes have been used to find modified documents and partial files of interest to investigators. They have also been used to find files such as photographs embedded in other files. Previous researchers have used fuzzy hashes at file-level to identify malware families and a combination of file and section hashes to confirm whether a file contains malware or to cluster malware by type (virus, worm, Trojan etc.).

Issues with file-level fuzzy hashing

Windows PE files share a structure of headers and details (sections) which in the header contents may be identified as being similar. The MS DOS header in a PE file usually contains the words “This program cannot be run in DOS mode” which will appear in the majority of PE files.

If a malware author inserts contrived redundant sections that are common to benign programs this will increase the similarity of the PE file to benign executables and decrease its similarity to other members of the malware family.

A second issue with similarity hashing at file-level is that in addition to content, the level of similarity depends on sequence. Executable file formats are highly structured, but it is a relatively trivial task for a malware author to insert redundant and unused sections into executables and to move sections from their normal position. Such interventions will not impact the loading and running of the software but will affect file-level similarity comparisons.

When testing executables for similarities (Pagani et al., 2018) found that relatively minor modifications to the source code such as swapping the sequence of instructions had a marked effect on file-level similarity reducing it to zero in some cases where 99.8% of the program was unchanged.

A common method used by malware authors to obfuscate their executables is to compress them using a wrapping packer utility such as UPX (UPX, 2017) or ASpack (What is ASpack?, 2018). These utilities “wrap” the compressed executable inside a small executable stub which decompresses the malware into memory and transfers control to it when executed. The compressed data is very different to the original unpacked version and will score close to zero when compared with a similarity hash. If two objects are compared in a packed and unpacked state, the similarity scores for the packed comparison are typically much lower than those of the unpacked comparison.

Section-level hashing involves using a third-party library function to split the Windows PE file into its headers and sections and applying a similarity hash function to each section. This paper presents a novel method of section-level hashing to overcome the limitations of file-level hashing. It calculates the ssdeep similarity hashes of each instance and section of a set of known malicious PE files and stores them in a database. File and section hashes are calculated for previously unseen malwares and compared to the set in the database for family classification. To reduce incorrect classifications, sections that are identical in multiple families are excluded from the comparisons.

Related work

Early research

Detecting similarity between files and other data objects has been a research topic that can be traced back at least to the early 1990s (Roussev et al., 2006). “Manber (1994) developed the “sif” tool, which seeks to identify file similarity based on approximate fingerprinting (essentially, selective hashing).” In 2002, Tridgell (2018) developed the “spamsum” algorithm to identify emails which were similar to, but not identical to known spam messages.

In (Ligh et al., 2010) the authors describe some use cases for similarity hashes including detection of shared code, finding related malware, detecting infections in memory dumps of multiple computers and comparing malware on disk to that in memory to identify self-modifying or metamorphic malware).

ssdeep – 2006

As described in (Kornblum, 2006), Kornblum developed an algorithm based on Trigdells' spamsum which he termed “Context Triggered Piecewise Hash” (CTPH) and implemented in a computer program which he called “ssdeep”. This was probably the first similarity hashing algorithm widely applied in computer forensics.

The algorithm combined piecewise hashing (where multiple hashes are generated for fixed-length input file segments) and a rolling hash (which maintains state based on the last few input bytes). In the CTPH, the fixed-length of the input file segments piece-wise hashing calculation is replaced with the rolling hash, when the rolling hash produces a pre-defined “trigger” value the piecewise hash is recorded and reset. The final output is the ssdeep hash value or “signature”' for the input file. The signatures for two files can be compared using ssdeep with a different command line option, to determine if they have a common origin (i.e. are similar). Similarity is measured on a linear percentage scale where 100 represents almost (possibly fully) identical and 0 represents no similarities. A weakness with ssdeep is that it cannot compare files and objects whose sizes are less than half or more than twice the size of the comparison file.

Other hashes and sdhash

Later research produced several other similarity hashes each claiming to perform better than ssdeep. This included MRS (2007) (Roussev et al., 2007), BitShred (2010) (Jang et al., 2010), bbhash (2012) (Frank and Baier, 2012), mvHash-B (Frank et al., 2013) and FirstByte (Upchurch and Zhou, 2013) in 2013. While BitShred was the first similarity algorithm specifically designed for malware triage, perhaps the best known and most widely implemented similarity hash other than ssdeep is “sdhash” (Roussev, 2010) developed in 2010. This incorporated the selection of statistically improbable features in binary files, screening out weak features and using Bloom filters to support the comparison of different sized objects.

Performance metrics

In 2015 (Upchurch and Zhou, 2015) presented a framework specifically for testing programs that used similarity comparisons to identify malware variants which they named "Variant". In addition to a fixed dataset, the authors proposed performance accuracy measures based on measuring true/false and positive/negative results which they termed Precision, Recall and F-Measure defined as follows:

- Precision: The fraction of retrieved samples that are classified correctly.
- Recall: The sensitivity of the algorithm to retrieve relevant instances.
- F-Measure: The harmonic mean of Precision and Recall (harmonic means are used in statistics in place of arithmetic means when dealing with averages of rates and ratios).

Effectiveness comparison

In 2016 (Forensic malware analysis, 2016), examined the effectiveness of four different similarity hashing implementations for identification of malware similarities in executable files previously classified by the VirusTotal platform. Using file-level ssdeep, sdhash, mvHash and mrsh v2 (Breitinger and Baier, 2012), they found that ssdeep was "suitable for matching similarities not only from malware of the same family but also from emerging families". They concluded that fuzzy hashing was an effective identification method and that it was capable of identifying new variants. Of the four algorithms tested, the authors presented sdhash as having the highest identification rate when used with a threshold of 21. In addition, only sdhash correctly identified a particular piece of malware as a variant.

Malware detection

In (Namanya et al., 2016) the authors studied the use of similarity hashing with Evidence Combinational Theory to calculate the similarity of Portable Executable files to ascertain whether a given PE file was malicious/not.

The authors used fuzzy logic and certainty factor models from Evidence Combinational Theory to calculate Common Factors (CFs) to assign a "degree of belief" weighting to each hash type.

Using a set of known malicious PE files, the researchers built a database containing ssdeep file and PE resource section hashes, ImpHash (a hash calculated from the PE file import section (Tracking Malware with Import Hashing, 2014)) and PeHash (a hash of the structural data in a PE file header and its sections (Wicherski, 2009))

They compared the database hashes from the known malicious files to those obtained from a second dataset containing both malicious and benign files, counting the true and false negatives and positives for each hash type. The counts were then used to calculate the Common Factors (CFs) for each hash type. The resultant CFs were normalized and used to weight similarity scores for each hash type in subsequent comparisons to calculate an overall malicious score (effectively a measure of how likely the file is to be malicious) for each file being examined which is presented to the analyst.

Their experiments showed that their combination method yielded overall recall (98.65%), accuracy (98.16%) and precision (99.33%) metrics which outperformed the results for individual hashes. These results were for the detection of malicious files, the experiments did not classify the files into malware families. The authors conclude that their method provides a means of detecting malicious PE files only needing dynamic analysis in cases where their system produces an inconclusive result.

Methodology

Experimental environment & sample data

The research experiments were conducted using specially developed Python application programs on Ubuntu 16.04 LTS. The Ubuntu machine was a VMware Workstation 12 Pro virtual machine with 6 GB RAM, 4 CPUs and 1.5 TB disk.

The March 2018 Windows 32 EXE and DLL PE file malware collection was downloaded from VirusTotal (VirusTotal private API, 2018). This comprised 96 GB in 69,632 executables, 5112 families and their associated VirusTotal classification data in JSON files. The JSON files included the details of the engines that analyzed the malware and their output family names.

Table 1 details the third-party libraries and utilities used.

The "malicia/AvClass" malware labelling tool (AVClass malware labeling tool, 2018) was used to extract the most likely family names from the VirusTotal JSON files. To increase the likelihood that the majority of VirusTotal engines agreed on the family name, only cases where 10 or more engines analyzed the executable and where at least 60% of those agreed on the family name were included. This reduced the dataset to 32,812 executables in 398 families. Each VirusTotal malware detection engine uses its own family naming conventions to group executables. The AVclass tool uses aliases to combine these different family names together. In addition, details of 1000 Windows 8.1 32-bit PE files were added to the dataset using a family name of "BENIGN".

Selection of malware families

Table 2 shows the four malware families used in the research experiments and the number of members in each family. The family names are those selected by the AvClass tool as described in the preceding section.

100 random non-obfuscated samples of each of the four families in the reference dataset and database were selected by calculating the entropy of the PE file. A compressed (packed) or encrypted PE file has a higher entropy than a non-obfuscated sample. Choosing the files with lower entropy scores therefore is more likely to yield non-obfuscated files. This was further verified by attempting to

Table 1
Libraries and utilities.

Name	Purpose
fuzzyhashlib	Generate ssdeep similarity hashes (Python hashlib-like wrapper for several fuzzy hash algorithms, 2018)
Pefile	Decompose PE file into sections (pefile, 2017)
malicia/AvClass	Mass malware labeler (AVClass malware labeling tool, 2018)
file_entropy	Calculate file entropy (Python Entropy calculator, 2010)
SQLite	Database library (About SQLite, 2018)
UPX/ASpack	Executable packers (UPX, 2017), (What is ASpack?, 2018)

Table 2
Malware families.

Name	Members
Virut	2936
Wabot	694
virlock	607
coinminer	139

pack the selected files and discarding any that failed.

100 PE files belonging to a mix of families excluding those in Table 2 were selected using the process detailed above. This mixed family set was used to measure classification performance for each experiment.

The effect of obfuscation was also evaluated by packing both the mixed family and 100 non-obfuscated sample sets and comparing them. Both sets were packed with UPX and ASpack packing tools.

Data collection

Data collection was performed using two Python programs. The first program generates and stores similarity hashes for the known executables. The second compares hashes of a sample of executables to those stored in the database to predict family membership.

Creation of the reference database

PE files are passed as input, their MD5 hash is generated and used to check that the file details are not already in the database. SHA256 and ssdeep hashes (Python hashlib-like wrapper for several fuzzy hash algorithms, 2018) are then generated. The hashes are subsequently stored in the "*FileDetails*" table in the database as shown in Table 3.

The "*ShortFamily*" and "*LongFamily*" fields are populated using the "*AvClass*" tool as described in section 3.1.

The files are then parsed into sections, using the "pefile" library (pefile, 2017). For each section, an MD5 and ssdeep hash are generated and stored in the "*Sections*" table, along with metadata in the form of section name and size in bytes. The record layout for this "*Sections*" table is shown in Table 4.

The MD5 hashes uniquely identify the parent PE file and the combination of PE file and section. This ensures that there are no duplicates. If a PE file has been previously examined and stored in the database, its MD5 can be used to report its family without the need to compute and compare the similarity hashes. (This latter function was disabled for the experiments).

Comparison of input PE file & section hashes to the reference database

The second Python program takes an individual PE file or a directory of PE files to be compared to the reference database as input. It calculates the MD5 and ssdeep hash values for the PE file and its sections, stores them in an array and compares these to the MD5 and ssdeep hash values stored in the reference database. The

Table 3
Database "FileDetails" table.

Field Name	Type	Purpose
FileMD5	Text	PE file MD5 hash
FileSHA256	Text	PE file SHA256 hash
FileSSdeep	Text	PE file ssdeep hash
ShortFamily	Text	Majority family name
LongFamily	Text	List of family names

Table 4
Database "Sections" table.

Field Name	Type	Purpose
FileMD5	Text	PE file MD5 hash
SectionMD5	Text	PE section MD5 hash
SectionName	Text	PE section name
SectionSize	Integer	Raw section size (bytes)
SectionSSdeep	Text	Section ssdeep hash

Table 5
Database "IgnoreSections" table.

Field Name	Type	Purpose
SectionMD5	Text	MD5 hash to ignore.

results can then be printed to standard output or re-directed to a text file. Section MD5 hashes that should be excluded from counting (e.g. those that occur in multiple families) may be added to the "*IgnoreSections*" table (Table 5).

The program allows the user to specify the ssdeep threshold (the value below which ssdeep comparison results are not reported) and whether to output the most likely family name or all family names of matches.

As section names in PE files may be renamed by a malware author, each section in the input PE file must be compared against all section records in the reference database. Considering only those hashes for sections that match an identically named input PE section would risk omitting highly similar sections with non-standard names.

Detailed explanation of comparison logic

At the file comparison level, the tool compares the ssdeep digest of each input file to the ssdeep digest of all other files on the database. All similarity scores above the threshold specified are counted and accumulated by malware family as recorded on the database. The tool selects the malware family with the highest number of similarity scores at or above the threshold as being the most likely family to which the input file belongs. The database record for the input file itself is excluded from the counts.

At the section level, the tool splits the input PE file into sections and compares the ssdeep similarity digest of each input section to that of all other sections in the database – ignoring input and database sections whose MD5 is in the "*IgnoreSections*" table. Each section record in the database contains the malware family of the parent PE file from which it was extracted when it was added to the database. All section similarity scores above a specified threshold are counted and accumulated by malware family as recorded on the database. The tool selects the malware family with the highest number of similarity scores above the threshold as being the most likely family to which the input section belongs. The family predictions for each section are counted and the family prediction for the input file is the family with the highest number of sections assigned to it. As with the file level comparisons, the database records for the input sections are excluded from the counts.

The "*IgnoreSections*" table is required as initial experiments showed that identical sections appear in many PE files across multiple malware families. Excluding these sections is needed to reduce the mis-classifications that would otherwise result. The "*IgnoreSections*" table is used in preference to removing the section details from the "Sections" table and as a fast means of identifying sections read from comparison PE files that should be ignored.

Performance metrics and calculations

The four possible outcomes in malware detection are explained as follows:

True Positive (TP): The method correctly identifies that a known malicious executable is malware.

True Negative (TN): The method correctly identifies that a known benign executable is not malware.

False Positive (FP): The method incorrectly identifies a known benign executable as malicious.

False Negative (FN): The method incorrectly identifies a known malicious executable as benign.

The performance accuracy measures proposed are calculated as:

Precision = TP / (TP + FP) - The ratio of correct positive to all positive (correct & incorrect) diagnoses.

Recall = TP / (TP + FN) - The ratio of correct positive diagnoses to correct positive and incorrect negative diagnoses (when the inputs are all malware)

F-Measure = 2 * (Precision * Recall) /(Precision + Recall).

Experiments

The 100 non-obfuscated PE files from each of the four families were used as input to the comparison tool at thresholds of 1, 30, 40, 50, 60 & 75 in turn and the results were recorded as follows:-

- True Positive: The tool correctly predicted the malware family of the input PE file
- False Negative: The tool incorrectly predicted that the input PE file was a member of a different family than it actually was.
- Unknown: The tool could not determine the malware family to which the input PE file belonged.

The 100 non-obfuscated mixed-family samples were used to provide true negatives and false positives. i.e.

- True Negative: The tool correctly predicted that the input PE file was NOT a member of the family.
- False Positive: The tool incorrectly predicted that the input PE file WAS a member of the family being examined when in fact, it was not.
- Unknown: The tool could not determine the malware family to which the input PE file belonged.

The four experiments were then repeated using the packed PE files from each family with the packed mixed-family sample and the results were recorded in the same manner.

Results

Non-obfuscated input files

For the sake of brevity, the file and section-level results for all 4 families are summed and presented in Tables 6 and 7. Optimal results are displayed in bold. Precision and Accuracy column headings are shortened to "Prec." and "Acc." respectively.

The minimum threshold is set to 1 to select only sections where the similarity score exceeds zero (no similarity).

In both cases, unknowns have been included with either false negatives or false positives as appropriate to provide a family vs-all-others comparison. The unknowns increase with threshold explaining the counter-intuitive increase of "FP" with threshold.

As can be seen from the tables, the section-level method performs much better than the file-level equivalent on all metrics and especially on recall.

Table 6
Non-obfuscated file-level ssdeep results.

T	TP	FN	TN	FP	Prec.	Recall	Acc.
1	**187**	**213**	**374**	**26**	**0.88**	**0.47**	**0.70**
30	182	218	374	26	0.88	0.46	0.70
40	164	236	370	30	0.85	0.41	0.67
50	139	261	370	30	0.82	0.35	0.64
60	120	280	366	34	0.78	0.30	0.61
75	97	303	360	40	0.71	0.24	0.57

Table 7
Non-obfuscated section level ssdeep results.

T	TP	FN	TN	FP	Prec.	Recall	Acc.
1	**358**	**42**	**396**	**4**	**0.99**	**0.90**	**0.94**
30	**358**	**42**	**395**	**5**	**0.99**	**0.90**	**0.94**
40	352	48	395	5	0.99	0.88	0.93
50	347	53	394	6	0.98	0.87	0.93
60	337	63	392	8	0.98	0.84	0.91
75	333	67	387	13	0.96	0.83	0.90

Obfuscated (packed) input files

As in the case of the non-obfuscated files, the file and section level results are summed and presented in Tables 8 and 9. Optimal results are shown in bold.

While it is evident that the metrics are not as favorable for either method in comparison to their non-obfuscated equivalents, when the file and section results are compared, the section method outperforms the file-level method by 1.89–2.6 times.

The results show that while packing does reduce the similarities in the files to the point where the file-level precision, recall and accuracy drop below 44% at all thresholds, the section level method remains above 68% at thresholds lower than 75. The packed section-method also returns superior recall and accuracy results than the non-obfuscated file-level method at thresholds of 60 and under.

Table 10 shows the calculated F-measures for each method.

(a) Non-obfuscated file.
(b) Non-obfuscated section.
(c) Obfuscated (packed) file.
(d) Obfuscated (packed) section.

The F-measure combines the precision and recall results giving a single metric (in this case where both are weighted equally). From Table 10 it can be clearly seen that the non-obfuscated section method (b) shown in bold is superior to the other methods while

Table 8
Packed file-level ssdeep results.

T	TP	FN	TN	FP	Prec.	Recall	Acc.
1	**171**	**229**	**120**	**280**	**0.38**	**0.43**	**0.36**
30	162	238	119	281	0.37	0.41	0.35
40	150	250	109	291	0.34	0.38	0.32
50	148	252	96	304	0.33	0.37	0.31
60	136	264	86	314	0.30	0.34	0.28
75	114	286	58	342	0.25	0.29	0.22

Table 9
Packed section-level ssdeep results.

T	TP	FN	TN	FP	Prec.	Recall	Acc.
1	**322**	**78**	**322**	**78**	**0.81**	**0.81**	**0.81**
30	320	80	322	78	0.80	0.80	0.80
40	317	83	309	91	0.78	0.79	0.78
50	317	83	295	105	0.75	0.79	0.77
60	323	77	256	144	0.69	0.81	0.72
75	232	168	182	218	0.52	0.58	0.52

Table 10
F-measures.

T	(a)	(b)	(c)	(d)
1	0.61	**0.94**	0.40	0.81
30	0.60	**0.94**	0.38	0.80
40	0.55	**0.93**	0.36	0.78
50	0.49	**0.92**	0.35	0.77
60	0.43	**0.90**	0.32	0.75
75	0.36	**0.89**	0.27	0.55

the obfuscated file method (c) returns the poorest results.

Conclusions & further work

The aim of this research was to improve the results obtained by file-level fuzzy hashing generation and comparison for malware family identification & classification. It has been achieved using section-level ssdeep hashing while ignoring sections that are common to multiple families.

The objective was to create a framework for calculating, storing, retrieving and comparing similarity digests, and to use this framework to conduct experiments comparing the two methods.

The experiments were conducted on a variety of malware types, families and variants and with non-obfuscated and packed malware. Using the ssdeep hashing algorithm they have shown that when unknowns are included as being an incorrect diagnosis, section-level digests are superior to file-level. When the best performing results of each method are compared, the section-level method returns 92% more true positives for non-obfuscated malware and 88% more for packed malware than the file-level method.

Results for packed malware can be improved by calculating its entropy to determine whether the file is packed, identifying which packer has been used with PEiD (PEiD, 2018) or VirusTotal, unpacking the PE file and using the unpacked file as input to the framework.

In terms of performance, when the measurement framework proposed by (Upchurch and Zhou, 2015) is used to compare the results of both methods, section-level digests significantly and consistently outperform file-level on precision, recall and accuracy metrics.

Optimal results for both packed and non-obfuscated malware were achieved when the ssdeep threshold level was set between 1 and 30. Results dis-improve at thresholds of 75 or higher for both types but more markedly so for packed malware.

A limitation of the section-level method is that it tends to produce more false negatives than the file-level method however these are compensated for when unknowns are included as false.

A further limitation is the speed of comparison. The initial version of the comparison tool requires reading every section record in the database to retrieve and compare its ssdeep hash to that of the input. Potential optimizations include loading the database into a numpy array and only comparing ssdeep hashes that are within a power of two of that of the input section.

An important discovery was that there are a lot of identical (as opposed to similar) PE sections which appear in many different malware families. These sections cause unwarranted similarity between malware variants and must be excluded from the prediction process to obtain accurate results. Two particularly prevalent examples are the "empty string" hash and the hash of the UPX0 section inserted by the UPX packer. In 1952 UPX0 section MD5 hashes, only 17 variants existed in the database. Furthermore, the raw size of the UPX0 segment was zero in 1715 of the executables. This adjustment is not possible with file-level analysis.

Some suggestions for future work:

- Optimizing the comparison tool as outlined above.
- Identifying particular sections which could be used to classify malware. This would have to go beyond using the section name for identification as these can be renamed. Such an optimization would reduce storage requirements and comparison times.
- Extending the framework to include the sdhash and tlsh algorithms and comparing the results to ssdeep. This could also include experiments to determine which algorithm is most resistant to deliberate attempts to frustrate similarity hashing (Baier and Breitinger, 2011).
- Broadening the work done by (Namanya et al., 2016) to include ssdeep hashes of sections other than the resource section, incorporating sdhash and tlsh digests and extending it to malware variant classification
- Investigating the method to classify encrypted malware.
- Incorporate supplementary sources of ground truth e.g. Malpedia (Malpedia).

Acknowledgements

Mr. Stephen Tomkin, fuzzyhashlib author for his assistance with using the library.

VirusTotal for providing access to sample malware PE files and their classification data.

References

About SQLite, 2018. URL https://www.sqlite.org/about.html. Date Retrieved 24 Aug 2018.

AVClass malware labeling tool, 19 Feb 2018. URL https://github.com/malicialab/avclass. Date Retrieved 5 Jun 2018.

Baier, H., Breitinger, F., 2011. Security aspects of piecewise hashing in computer forensics. In: IT Security Incident Management and IT Forensics (IMF), 2011 Sixth International Conference on, pp. 21–36.

Breitinger, F., Baier, H., 2012. Similarity preserving hashing: eligible properties and a new algorithm mrsh-V2. In: International Conference on Digital Forensics and Cyber Crime, pp. 167–182.

Forensic Malware Analysis: the Value of Fuzzy Hashing Algorithms in Identifying Similarities, 2016. IEEE Trustcom/BigDataSE/ISPA, Trustcom/BigDataSE/ISPA, 2016 IEEE, TRUSTCOM-BIGDATASE-ISPA, p. 1782, 2016.

Frank, Breitinger, Baier, Harald, 2012. A fuzzy hashing approach based on random sequences and hamming distance. In: Proceedings of the Conference on Digital Forensics, Security and Law. Association of Digital Forensics, Security and Law, p. 89.

Frank, Breitinger, Asteb, Knut Petter, Baier, Harald, Busch, Christoph, 2013. mvhash-b-a new approach for similarity preserving hashing. In: IT Security Incident Management and IT Forensics (IMF), 2013 Seventh International Conference on. IEEE, pp. 33–44.

Jang, Jiyong, Brumley, David, Venkataraman, Shobha, 2010. Bitshred: Fast, Scalable Malware Triage. *Cylab*, vol. 22. Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU-Cylab-10.

Kornblum, Jesse, 2006. Identifying almost identical files using context triggered piecewise hashing. Digit. Invest. 3, 91–97.

Ligh, Michael, Adair, Steven, Blake, Hartstein, Richard, Matthew, 2010. Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code. Wiley Publishing.

Malpedia. URL https://malpedia.caad.fkie.fraunhofer.de/.

Namanya, Anitta Patience, Ali Mirza, Qublai Khan, Mohannadi, Hamad Al, Awan, Irfan U., Disso, Jules Ferdinand Pagna, 2016. Detection of malicious portable executables using evidence combinational theory with fuzzy hashing. In: Future Bibliography 99 Internet of Things and Cloud (FiCloud), 2016 IEEE 4th

International Conference on. IEEE, pp. 91–98.

Pagani, F., Dell'Amico, M., Balzarotti, D., 2018. 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, pp. 354–365.

pefile 5 Nov 2017. URL https://github.com/erocarrera/pefile. Date Retreived 31 May 2018.

PEiD, 24 Apr 2018. https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml. (Accessed 24 August 2018).

Python Entropy calculator, 29 Nov 2010. URL https://github.com/ActiveState/code/tree/3b27230f418b714bc9a0f897cb8ea189c3515e99. Date Retrieved 26 Jul 2018.

Python hashlib-like wrapper for several fuzzy hash algorithms, 3 June 2018. URL https://github.com/sptonkin/fuzzyhashlib. Date Retrieved 31 May 2018.

Roussev, Vassil, 2010. Data fingerprinting with similarity digests. In: IFIP International Conference on Digital Forensics. Springer, pp. 207–226.

Roussev, Vassil, Chen, Yixin, Bourg, Timothy, Richard III, G., 2006. md5bloom: forensic filesystem hashing revisited. Digit. Invest. 3, 82–90.

Roussev, Vassil, Richard III, G., Marziale, Lodovico, 2007. Multi-resolution similarity hashing. Digit. Invest. 4, 105–113.

Symantec ISTR 2018, 2018. URL https://www.symantec.com/security-center/threat-report. Date Retrieved 30 Mar 2018.

Tracking Malware with Import Hashing, 23 Jan 2014. URL https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html/ Date Retrieved 10 Oct 2018.

Tridgell, A.. Spamsum. http://samba.org/ftp/unpacked/junkcode/spamsum/README. (Accessed 4 April 2018).

Upchurch, Jason, Zhou, Xiaobo, 2013. First byte: force-based clustering of fitered block n-grams to detect code reuse in malicious software. In: Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on. IEEE, pp. 68–76.

Upchurch, Jason, Zhou, Xiaobo, 2015. Variant: a malware similarity testing framework. In: 2015 10th International Conference on Malicious Unwanted Software (MALWARE), p. 31.

UPX, 12 May 2017. The Ultimate Packer for eXecutables. https://upx.github.io/. (Accessed 24 August 2018).

VirusTotal private API. URL https://www.virustotal.com/en/documentation/private-api Date Retrieved 30 Mar 2018.

What is ASpack?, 2018. URL http://www.aspack.com/aspack.html. Date Retrieved 24 Aug 2018.

Wicherski, Georg, 2009. peHash: a novel approach to fast malware clustering. In: LEET'09 Proceedings of the 2nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More, vols. 1–1.