Detection of Malicious Portable Executables using Evidence Combinational Theory with Fuzzy Hashing.

Anitta Patience Namanya[1] , Qublai Khan Ali Mirza[1],
Hamad Al-Mohannadi[1], Irfan U. Awan[2]
School of Electrical Engineering and Computer Science,
University of Bradford.
Bradford, UK
[1]{A.Namanya, Q.K.AliMirza, H.I.M.AL-Mohannadi}@student.bradford.ac.uk
[2]I.U.Awan@bradford.ac.uk

Jules Ferdinand Pagna Disso
Research and Development,
Nettitude Intelligent Cyber Security and Risk Management.
Leamington Spa, UK
jpagnadisso@nettitude.com

***Abstract*— Fuzzy hashing is a known technique that has been adopted to speed up malware analysis processes. However, Hashing has not been fully implemented for malware detection because it can easily be evaded by applying a simple obfuscation technique such as packing. This challenge has limited the usage of hashing to triaging of the samples based on the percentage of similarity between the known and unknown. In this paper, we explore the different ways fuzzy hashing can be used to detect similarities in a file by investigating particular hashes of interest. Each hashing method produces independent but related interesting results which are presented herein. We further investigate combination techniques that can be used to improve the detection rates in hashing methods. Two such evidence combination theory based methods are applied in this work in order propose a novel way of combining the results achieved from different hashing algorithms. This study focuses on file and section Ssdeep hashing, PeHash and Imphash techniques to calculate the similarity of the Portable Executable files. Our results show that the detection rates are improved when evidence combination techniques are used.**

Keywords: Malware detection, Fuzzy hash, Evidence combinational theory, Common Factor Model, Fuzzy Logic, Portable executable.

I. Introduction and Motivation

The vast number of malware samples collected from the wild does not allow for full analysis of each sample discovered. For instance, by the end of 2015, nearly 500 million malware were collected according to McAfee and Av-Test Institute [1, 2] of which about 145 million were new samples. With such statistics, there is a need to create methods that shorten the analysis process by allowing isolation of files which are variations of samples already known. This is why most of the computer security firms are calling for an integrated approach towards malware detection and have actually developed systems that detect malware in stages [3]. The initial stage of triaging the malware in order to cluster the samples is a well-known step that normally uses hashing as a methodology. Although hashing faces the issue of high false negatives, a combinational approach could lead to better results. By focusing on files of the same type, structure as a discerning factor is eliminated. Since more than 90% of all computer users still use windows operating systems [4], this work focuses on Portable Executable files. The similarity in the files detected by the hashing functions is used as the content similarity factor for the sample of the dataset used. Many reasoning models based on uncertainty have been developed to enable expert systems to make decisions based on unreliable data [5]. The novel ideas in this work are:

- The use of evidence combinational theory to combine the results from different hashing methods.
- The investigation into the effect of the application of evidence combinational theory to the hashing analysis.
- A quantifiable metric to represent the level of maliciousness of a file for a system user.

This paper has been organised as follows; the first section introduces the work providing the motivation and the contributions of the study. The second section, discusses the background by defining the different hashes and combination methods used. The third section provides the related work and the identified gaps this study covers. Section 4 describes the design approach, the steps taken to model the system architecture and the different algorithms used in the implementation. We explore the evaluation methods used for this model in section 5 and present our results at the different stages of the system design in the section 6. The analysis of the achieved results is section 7 and the achievements and limitations of the proposed technique are discussed in section 8. Finally, we conclude our work and present the future work in section 9.

II. Background

Presently, malware detection heavily relies on the expertise of malware analysts to build a system to filter out malicious files while making sure that the ones that a user needs can still work within the system. The analyst dissects a discovered suspicious sample and designs signatures or systems to detect future instances of the sample and thereafter recovery if possible. There is a dearth of such expertise to handle the vast amounts of new malicious discoveries. This means that new automated

978-1-5090-4052-0/16 $31.00 © 2016 IEEE
DOI 10.1109/FiCloud.2016.21

methodologies that make decisions based on uncertain data from the used analysis methods are required. Most expert systems show low errors in decisions that are based on uncertainty because of the different mathematical theories developed and implemented [5]. File hashes are some of the information retrieved from the file during static analysis. In this section, different hashing techniques that are used in malware analysis and the different reasoning under uncertainty methods that have been identified for this specific study are discussed.

A. *Hashing algorithms*

Hashing algorithms compute digests of a file that are unique to the sequence of the file binary contents and structure [7].

1) The Cryptographic Hashes like MD5, SHA1 and SHA256 have been very popular in malware identification since they are commonly used to check for the integrity of files. While this is a safety measure that works for executable files where care is taken during file installation, sometimes, files are downloaded and installed in a hurry or without the knowledge of the computer user. Cryptographic file hashes might not be very useful in such cases since a simple bit change in the file affects the hash digest computed. They have, however, been repurposed and become useful in other methods. For example, section's MD5 and SHA hashes can be used to detect types of packing and in some cases classification of malware families [7]. For the purpose of this study we do not use these specific hashes, we however, focus on fuzzy hashing and feature specific file and section hashes that are further discussed in detail in the next subsections.

Ssdeep Hash: Also known as Context Triggered Piecewise Hashing(CTPH) [9] or fuzzy hashing, Ssdeep was first introduced in anti-spam research to detect similarity in files. The original idea called Spamsum was developed by combining the piecewise hashing (Fowler/Noll/Vo –FNV hash) and the rolling hashing to produce a Non cryptographic hash that is then used by a comparison algorithm that uses Levenshtein Distance to compare any 2 generated hashes for sequence similarity [6, 11]. It was later adopted by Kornblum [9] into Ssdeep for the purposes of forensic science and this application was extended to malware by the Mandiant cybersecurity firm with the purpose of providing the malware analysts [11] with information to guide their next step in the file analysis. Ssdeep is an open source program that generates fuzzy hashes that when two are compared, a similarity percentage score between the files is returned with a very high confidence of 99. It is now a very crucial step in static analysis with many analysis tools attaching this hash next to the cryptographic hashes in any malware analysis report. This string is used to provide the similarity percentage [0-100] when compared with another hash from another file. The percentage of similarity attached to any two files can sometimes justify why the files are the same or in the same family of malware. In this work, the file Ssdeep hash and the resource section Ssdeep hash are considered.

2) Imphash: First proposed by Mandiant cybersecurity firm [22], Imphash is a hashing method that is calculated from the digest of the import section of the executable file. With many researchers focusing on the imported APIs as a way of understanding how the file would interact with the system based, the Import table which holds the APIs under traditional conditions provides added insight into the expected behaviour of the file. Imphash matching allows the analyst to cluster malware based on the order and the contents of the import table [21]. This means that a change of order in the imports in table compromises this hash value and packers can also be used to overcome this hash as a detection method since the import table is sometimes hidden. However, Imphash is still very useful considering that most malware normally have a common origin with collaborating information which allows for clustering and detection of similar structured malware. Imphash has been incorporated in many static analysis tools like VirusTotal.com, Peframe and Pefile among others.

3) PeHash is a function that generates the binary cryptographic hash value of the structural data found in the file header and executable's section data. It is developed in [13] and its analysis shows that this metric hash provides good clustering matches for instances of similar polymorphic malware samples. PeHash has not been fully extended into most static analysis tools since it is file type specific however since this work focuses on PE type files, this hash is considered.

B. *Evidence Combinational Methods.*

These are methods used for decision making by combining results from different support methods [5] which have a degree of uncertainty. In malware analysis, analysts are more likely to obtain uncertain information from different analysis methods. Based on their expertise, they make decisions on the maliciousness of the file. Evidence combinational methods provide a way to automate this process [16]. If we have two different pieces of evidence with two different degree of belief (**X** with Degree x and **Y** with degree y) that support the hypothesis (***M***) that the file is either malicious or not, the result heavily depends on the degree of belief placed on the different pieces of evidence. Using the fuzzy set union operators T-conorms introduces logical connectives to design the reasoning system [14] based on the degrees of belief. We use strict Archimedean t-conorm because they can approximate every continuous t-conorm that take the value in the range (0-1) [14]. The two identified methods; fuzzy logic and the certainty-factor model are discussed below.

1) Fuzzy logic: Fuzzy logic is used in decision making where there is no deterministic data. The theory based on fuzzy sets was introduced in 1965 [15] and the resultant fuzzy logic approach follows that the end result is only true if and only if

either of the support evidence is true. Considering the initial hypothesis of Maliciousness (**M**), our degree in belief of this hypothesis using fuzzy logic approach defines the function:

$$x * y \ in \ \boldsymbol{M} \qquad (1)$$

Using the important class of "strictly Archimedean" t-conorms of fuzzy logic [16] the algebraic sum is given by:

$$x * y = x + y - x \cdot y \qquad (2)$$

This is the sum used to assign a new percentage belief on the overall degree of belief in the hypothesis.

2) The certainty Factor model (CF model): A reasoning method that manages uncertainty in rule based systems which was developed in 1975 for MYCIN expert system [17]. MYCIN was a rule based expert system that was designed to diagnosise baterial infections [6]. To compute the overall belief in the hypothesis, an expert represents the uncertainty in the rule by using a single Common Factor (CF) for every rule. The CFs work as the degree of belief attached to each rule. Following the T-conorms, the degree of belief in the overall hypothesis using two supporting evidence is always higher or equal to the degree of belief in M obtained from one piece of evidence [18]. Using the notation defined previously, the overall belief is given by:

$$x * y = \frac{x + y}{1 + x \cdot y} \qquad (3)$$

III. Related work

Hashing functions are used for malware clustering and triaging in many malware analysis scenarios. Since there is predefined meaning to the file similarity achieved between two malicious files, the interpretation of the fuzzy hashing score is better left to the deduction skills of the analyst. Many studies have been conducted on the clustering sensitivity of hashing methods. A section of a technical report by DigitalNinjas [19] provides an initial study of fuzzy hashing similarity. The work shows promise for the detection of the different families of malware that provides a level of confidence of 67% using only fuzzy hash technique. Although this work serves as a baseline, there is no comparison study against clean files as a control. This work is further extended by French and Casey [20] who carry out a study on the different fuzzy hashing methods available, Ssdeep and Sdhash. Although this work provides insight into how and why fuzzy hashing works, it still focuses on its use in clustering. It validates the use of hashing in clustering families of malware and calls for a cost benefit analysis of the hashing methods. Arik, et al [21] proposes a methodology to use Imphash for malware clustering which was first introduced by Mandiant in [22]. New fuzzy hashing methods are introduced in [6] as new clustering algorithms that provide higher sensitivity rates for the clusters. Although the results show higher sensitivity matching, hashing is still restricted to clustering of the malware samples. Furthermore, [6, 8, 12, 19-23] all use one hash to study each clustering experiment without looking at the potential increase in detection rates one could achieve by combining different hashing methods. This study focuses on investigating the effect of combining all the similarity hash methods for malware detection purposes and how this can be achieved by using mathematical theories rooted in uncertainty reasoning. The fact that most malware authors tend to use the same tools enables us to look at the possibility of similarity of the different sections and the files as a measure of maliciousness.

IV. Approach Design and System Architecture

In this section, we present the approach used in this study and describe the design and implementation of the algorithms that form the proposed method of malware detection. While designing the methodology for this study, the PE format and the work in [24] are revisited. By focusing on only the hashing aspect of static analysis, we investigated ways in which the different hashes that define PE fields of interest can be used to detect the malicious samples in the dataset. Table I shows the notations used in the design of the different algorithm with the approach. The methodology is presented in Fig 1 and divided into 5 different Steps as discussed next.

TABLE I. THE ALGORITHM NOTATIONS

Notation	Meaning
DB	Database
ImpH	Imphash
PeH	PeHash
FuzH	File Ssdeep Hash
ResFH	Resource Section Ssdeep Hash
Xi	Set of elements of i attribute
MD5	MD5 sum
CFH(a, b)	Ssdeep Hash Comparison Function to detect similarity percentage of a and b hashes.
HashFlag_set (H)	The Hash flag setting function for H type of hash
π_i(DB)	All the tuples in DB of attribute i
$\sigma_{a=b}(DB)$	Generalised selection of all tuples in DB where a= b
Pop_MASHDB	Populate Malware Sample Hash Database Function
MALHACompare(*f*)	Malware Hash Compare Function for file *f*
Mal (*f*)	Malicious Measure of File *f*
i	Hash of type i where: i = {1, 2,3,4} ⇔ { ImpH, PeH, FuzH, ResF}
FST	Fuzzy Logic Combinational Metric
CFM	Certainity Factor model Combinational Metric
TDR	True Detection Rate
CFi	Common Factor of Attribute i
SFi	Evidence Support Factor of Attribute i
FuzzyLogicSum	The Fuzzy Logic Algebraic sum function
MYCINSum	The Common Factor (MYCIN) Algebraic sum function

Step 1: Single File hashing Study

This was the initial study done on one clean file (arp.exe) found in Windows systems. Then the file was edited using

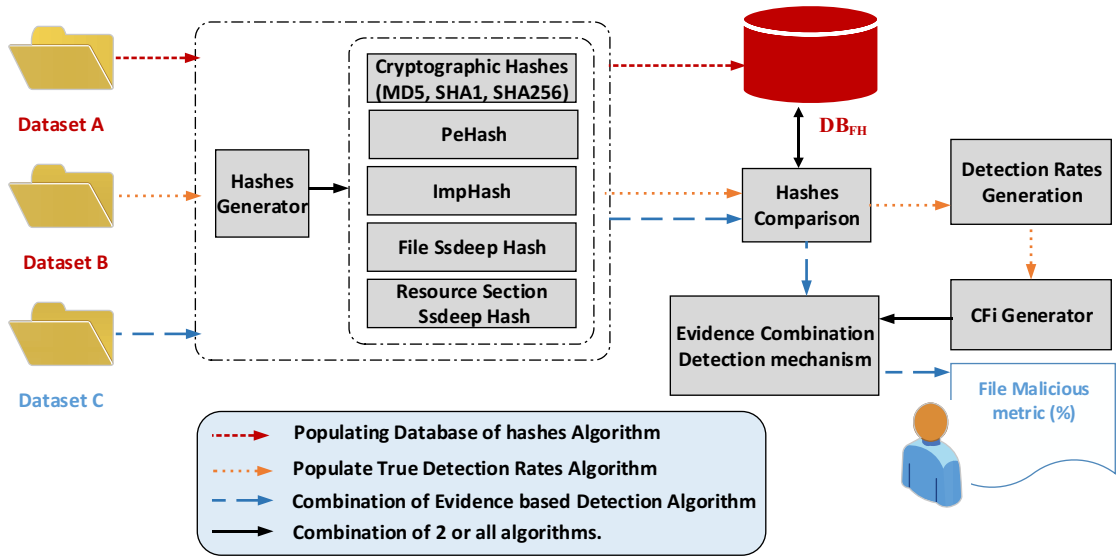

Fig 1. The pictorial representation showing how the algorithms interact with the elements of the system design.

Radare [25] to write "?a" characters into the file as saved with a different name. The hashes from the two files; the original and the edited version were compared so that the differences would provide a baseline for the next phase of the study.

Step 2: Collecting the Datasets

Collecting the datasets in Table II involved:

a) Downloading malware from different online repositories, using our own honeypots over a period of time and downloading from the malware repository at Nettitude, UK to collect a sample of malicious PE files for this study. The percentage formulation of the malware families of the malware dataset are shown in Table III.

b) Installing clean virtual machine images of Windows OS and running a batch script to collect all .exe files. For this specific study, we collected files from both the Windows XP and Win 7 environments.

TABLE II. THE EXPERIMENT DATASET

File Status	Total Files
Malicious files	21748
Clean files	1240

TABLE III. FAMILIES DISTRIBUTION IN THE MALWARE DATASET

Malware Family	Percentage
Adware	71.89%
Trojan	12.95%
Dropper	2.52%
Worm	12.47%
Spyware	0.08%
Downloader	0.07%
Virus	0.01%
Exploit	0.02%

TABLE IV. DATASETS FORMATION AND THEIR USES

Dataset	Number of files	Use in the system
A	7124	Training phase.
B ⟵{Bm, Bc}	7978	Baseline-Creation phase and CF generation.
C ⟵{Cm, Cc}	7886	Detection-method evaluation phase.

The files in the dataset were clearly labelled clean or malicious and were kept in separate folders depending on the known status of the files. The malicious files dataset is split into 3 different subsets with the sets; A, Bm and Cm and the Clean files into 2 subsets; Bc and Cc which were used for different steps as shown in Table IV.

Step 3: Training Database of Hashes

For our system to have a baseline, the randomly selected malware samples in dataset A are used to populate the database of hashes (DBFH). Algorithm I describes the process used to generate the database of hashes. The hashes of each executable file; MD5, Imphash, PeHash, Ssdeep Hash and Resource section Ssdeep Hash are computed and then saved into an SQLite database. This database of hashes (DBFH) is used in the next phases of the study as detection signatures.

ALGORITHM I: ALGORITHM FOR POPULATING DATABASE OF HASHES

Input: Malware Dataset *A*
Output: Flagged Hashes Database DB_{FH}

```
1: procedure: Pop_MSHDB
2: for malware m in A do
3:     Extract the file hashes
4:         Hashes (m) ⟵ {MD5, ImpH, PeH, FuzH, ResF}
5:     If Hashes(m) ∉ D then
6:         add Hashes to DBFH
7: end for
8: end procedure
```

Step 4: Malware Detection study

Once the training database was populated, the next phase involved comparing the file hashes of each file in the dataset against DBFH to determine how the different indicators would work with respect to detection rates. As a control measure, the clean files database was also tested against DBFH. The purpose was to collect the HashFlag_set based on which hash returned a positive response for the hashes of each file in the Dataset B. In order to do this, the 4 respective hashes of each executable file in Dataset B are computed and then the hashes are used as query variables to query DBFH. The Hash's corresponding flag is set if and only if it returns a positive response. MD5 hash is ignored in this algorithm because the file MD5 gives us absolute certainty that the files are similar.

Since the files were tagged clean or malicious depending upon the status, false and true detection could easily be identified based on the confusion matrix approach.

ALGORITHM II: ALGORITHM FOR POPULATING DETECTION RATES

Input: ds ⟵B , DB_{FH}
Output: DetectionRates

Overall Hash Based Detection Rate Phase

procedure: HaBaDR
for file (f) **in** B **do**
  H_flagset $_f$
  for i = 1 → 4 ▷Loop through the hashes set ids
    if f ∈ B **then**
      if H_flagset $_{fi}$ **then**
        TP_i = +1
      else
        FN_i = +1
      end if
    end if
    if f ∈ *Clds* **then**
      if H_flagset $_{fi}$ **then**
        FP_i = +1
      else
        TN_i = +1
      end if
    end if
    Update DetectionRates$_i$ ⟵{TP_i, FN_i, FP_i, TN_i}
  end for
return DetectionRates
end procedure

The detection rates across all the hashing methods that are obtained from Algorithm II where for the known malicious files, the presence of a set flag is counted as a TP and lack of one means a FN for the specified hash type. For the clean files, presence of a set flag is counted as a FP while lack of the flag means a TN count. The results from the count are normalised to be used as the belief factors in the evidence combinational approach implementation.

Step 5: Evidence Combination Theory Approach

The detection rates obtained in phase 4 are used as a measure of belief for each hash detection factor. True detection rates are used as a measure of the hash methodology accuracy. In order to make the detection rates compatible with the combinational theories, the positive detection rates are normalised to probabilities that add up to 1. The normalised detection rates take the form of the degree of belief in the uniform range [0, 1].

ALGORITHM III: COMBINATION OF HASHES BASED DETECTION MECHANISM

Input: PE file f, CF and DB_{FH}
Output: Mal(f)

Extract file Hashes Phase

procedure:HaBCoMalD(f)
Extract
  Hashes(f) ⟵{MD5, ImpH, PeH, FuzH, ResFH}
return Hashes(f)

MD5 Comparison Phase

$MD5_f$ ⟵ Hashes(f) $_{MD5}$
X_{MD5} ⟵ $\leftarrow \pi_{MD5}\left(\sigma_{"MD5=MD5_f"}\left(DB_{FH}\right)\right)$
if $X_{MD5} \neq$ *null* ***then***
  Mal(f) = 100%
  end procedure
else
  goto *HashComp*
end if
return *Mal(f)*
end procedure

Hashes Comparison Phase

HashComp:
  $ImpH_f$ ⟵ Hashes(f) $_{ImpH}$
  PeH_f ⟵ Hashes(f) $_{PeH}$
  $FuzH_f$ ⟵ Hashes(f) $_{FuzH}$
  $ResFH_f$ ⟵ Hashes(f) $_{ResFH}$
$X_{ImpH} \leftarrow \pi_{MD5}\left(\sigma_{"\mathrm{Im}\,pH=\mathrm{Im}\,pH_f"}\left(DB_{FH}\right)\right)$
  if $X_{ImpH} \neq$ null **then**
    $SF_{impH} = CF_{impH}$ * 1.0
  else
    SF_{impH} = 0
  end if
$X_{PeH} \leftarrow \pi_{MD5}\left(\sigma_{"PeH=PeH_f"}\left(DB_{FH}\right)\right)$
  if $X_{PeH} \neq$ null **then**
    $SF_{PeH} = CF_{PeH}$ * 1.0
  else
    SF_{PeH} = 0
  end if
$X_{FuzH} \leftarrow \pi_{MD5,FuzH}\left(DB_{FH}\right)$
  if $X_{FuzH} \neq$ null **then**

    $SF_{FuzH} = CF_{FuzH}$ * (Max (CFH ($FuzH_f$, $\forall x_1$,({x_0, x_1} ∈ X_{FuzH}))))
  else
    SF_{ResFH} = 0
  end if
$X_{ResFH} \leftarrow \pi_{MD5,ResFH}\left(DB_{FH}\right)$
  if $X_{ResFH} \neq$ null **then**
    SF_{ResFH} =CF_{ResFH} * (Max (CFH ($ResFH_f$, $\forall x_1$, ({x_0, x_1} ∈ X_{ResFH}))))
  else
    SF_{ResFH} = 0
  end if
$Mal(f)_{FST}$ = **FuzzyLogicSum(**SF_{PeH} , SF_{impH} , SF_{ResFH}, SF_{ResFH})* 100%
$Mal(f)_{CFM}$ = **MYCINSum(**SF_{PeH} , SF_{impH} , SF_{ResFH}, SF_{ResFH})* 100%
end procedure

The degree of belief/ Common factor for each Hash method (CFi) is defined as:

$$CF_i = \left[\sum_{n=1}^{4} TDR_n\right]^{-1} * TDR_i \qquad (4)$$

$$\text{Where } TDR_a = \frac{TP_a + TN_a}{Number\_in\_DatasetB}$$

The CF values obtained are inputs to Algorithm III for the combinational approach in the experiment. In this algorithm, the 4 hashes of the file under analysis are calculated and then used to query the DBFH.

For both the Imphash and PeHash, a true return of query means that the support factor (SF) for the hash is equal to the common factor (CF) or it is set to zero otherwise. For the file Ssdeep hash and resource section Ssdeep hash, the query returns the file which returns the maximum percentage of similarity. This percentage is then multiplied with the CF of the hash to get the utilised SF of the Hash.

The Malicious Score of the file is calculated from combining all the SF values of the receptive hashes using equations 2 and 3. This score is what is presented to the user to influence their decision making so the system allows the user to have the final decision that is well informed based on the metric achieved.

V. Test Environment

The test environment uses the tools specified in the Table V. We use a Linux-based operating system because the files under analysis are .exe files. The algorithms are scripted in python and some functions from Peframe [26] and Pefile [27] are extended in order to compute the file hashes under consideration; Ssdeep, PeHash, and Imphash and the database of Hashes is SQLite managed.

TABLE V. TEST BENCH SPECIFICATIONS

Tool	Specifications/ Details
Computer	Dell T1700, CPU – Intel Xeon@ 3.1GHz, RAM 32GB. Hard Disk – 500GB
Machine OS	Linux Mint 17.1 (#64 – Ubuntu SMP)

VI. Evaluation of the Proposed Method

While preparing the dataset, the malicious files and the clean files were well labelled so that the analysis of the results would follow the confusion matrix. The options in the confusion matrix in describe the different detection rates in the system and metrics are defined as [28]:

True Positive (TP): files that form part of the malicious dataset that are identified by the method as malicious.

False Negative (FN): files that form part of the malicious dataset that not flagged as malicious by the method.

True Negative (TN): files that form part of the clean files dataset that not flagged as malicious by the method.

False Positive (FP): files that form part of the clean files dataset that are flagged as malicious by the method.

Recall: a measure of the actual positive samples detected.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5)$$

Precision/ Positive Predictive Value (PPV): a measure of the actual positive samples for all the positive detections.

$$\text{Precision/ PPV} = \frac{TP}{TP + FP} \quad (6)$$

Accuracy: a measure of the true detections.

$$\text{Accuracy (ACC)} = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

VII. Experiementation Results and Analysis

Herein, we present the results achieved in the study. The results from the single file test shown in Table VII validate the theory that a small change in a file can greatly affect the results of similarity matching in some hashes while having very little to no effect in others. This justifies the investigations further carried out in this study on the hashes that return a similarity greater than 0%.

In order to analyse the logs obtained from the experiments, the results are normalised so that there is 0% detection for the "unknown" detection results. The graphs in Fig. 2 show how various hashes stack up against each other in detection rates.

TABLE VI. RESULTS FROM THE STEP 4 OF THE STUDY

		ImpH	PeH	FuzH	ResFH
Detection Rates	FP	0.06	0	0	0.01
	TN	0.94	1	1	0.99
	TP	0.97	0.96	0.95	0.89
	FN	0.03	0.04	0.05	0.11
Recall (%)		97.2	96	95.21	88.97
PPV (%)		99.4	100	100	99.89
ACC (%)		96.9	96.35	95.61	89.85
Detection Rates	TRUE (%)	96.9	96.35	95.63	89.85
	FALSE (%)	3.1	3.65	4.38	10.15
CF		0.256	0.254	0.253	0.237

TABLE VII. COMPARISON OF HASHES FROM THE SINGLE FILE STUDY

Hash Type	Original File Value	Edited File Value	Similarity match(%)
MD5	33f9b0e02d9d93f920605d02fb53f3fd	accd6591b8b8dad5f7f1470c90971e75	0
SHA1	4a22e401ad5adb7b3de8f819e86d8461d764d195	06b98e35c1f92f844b57376ee467ee977cc074bd	0
SHA256	1f4c090dfa389b3c6b16eb42299fb815f24efac7ca541bb6 0821e3da0131b8f6	bd4f056223439e83f2fffbe3c463e178da8465fabeb5124 3c04a3d2922de8fa2	0
Ssdeep-File	384:5u3Smmq6aYaBpYFAfjhXrToHWS4mW4sme9V: Avmq6affYFAfjhr8sgE	384:5u3Smmq6aYaBpYFmfjhXrToHWS4mW4sme9V :Avmq6affYFmfjhr8sgE	99
PeHash	5515f8e47661c7e170aee948cca7c8dc6198c08f	5515f8e47661c7e170aee948cca7c8dc6198c08f	100
Imphash	880bb6799a6e1a5ff7b4f022ff4003a9	880bb6799a6e1a5ff7b4f022ff4003a9	100
Ssdeep – Resources section	96:8EWS1pEmWwOh/VsBgtAb88caS5Ur9I5fa9VWPB MXsmrC9V:NWS4mWNJXCu6Xsme9V	96:8EWS1pEmWwOh/VsBgtAb88caS5Ur9I5fa9VWP BMXsmrC9V:NWS4mWNJXCu6Xsme9V	100

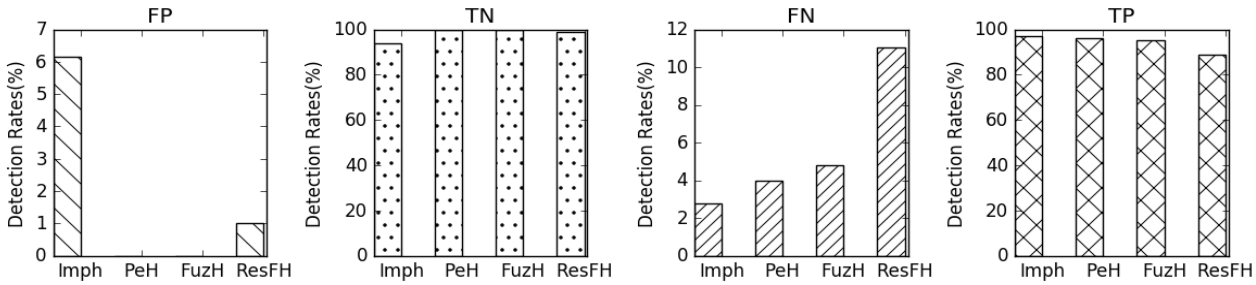

Fig. 2. The Individual Hashes Detection Rates using Dataset B for False Positive (FN), True Negative (TN), False Negative (FN) and True Positive (TP) cases receptively .

TABLE VIII. RESULTS FROM THE STEP 5 OF THE STUDY

	Detection Rates		Recall (%)	PPV (%)	ACC (%)
	True (%)	False (%)			
ImpH	97.24	2.76	97.57	99.39	97.24
PeH	96.29	3.72	95.92	100	96.29
FuzH	96.11	3.89	95.73	100	96.11
ResFH	90.39	9.61	89.55	99.89	90.39
CHM	98.16	1.84	98.65	99.33	98.16

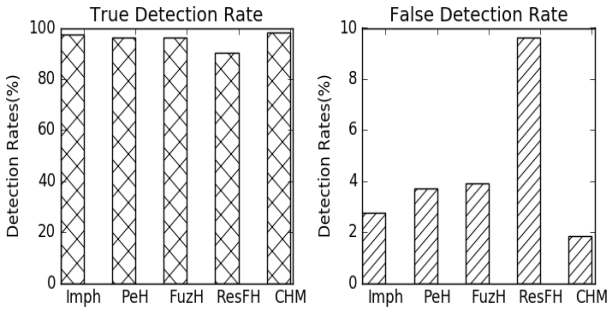

Fig. 3: True and False Detection Rates comparison for the individual Hashing Algorithm and Combined Hashing Methodology.

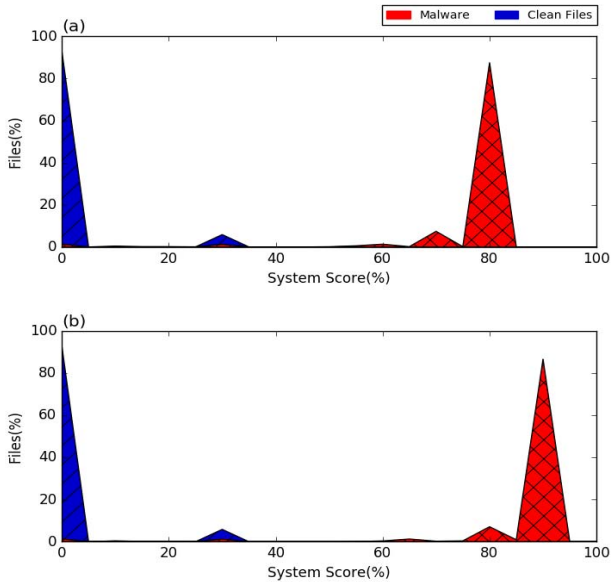

Fig. 4. The Combined Hash Score Clean and Malware File Area curves (a) Fuzzy Logic Method and (b) Common Factor Method.

Dataset B was used to populate the common factors (CF) as shown in Table VI. Ensuring that only the percentage of time that the hashing detection is right is utilised to calculate CF reduces the margin of error introduced in the next stages of the implementation. The calculated CF are used in Algorithm III to provide the overall file malicious percentage against PE files in dataset C. Table VIII and Fig. 3 compares the results achieved for the hashing methods and the proposed method. The proposed technique achieves an overall true detection rate of 98.2% and a false detection rate of 1.8% higher than any of the hashes.

In order to further understand how the malware file scores match up against the clean file scores which are shown by the area curves shown in Fig. 4 were drawn. Each method used is investigated independently. Both methods have over 99% of the clean file scoring well below 50% while over 97% of the known malware files score above 50% in either of the systems.

VIII. EVIDENCE COMBINATION OF HASHES FOR MALWARE DETECTION

Different hashes are introduced in [6, 11, 12, 21] and analysed to validate their high sensitivity in classification of malware families. The results achieved of 99.89% precision for the introduced resource hash provides a strong argument for this section hash in similarity matching. Although the results in [20] show that Ssdeep is the better performing of Sdhash and Ssdeep across all malware families, as malware get more complex, family clustering might become more tedious. This study successfully introduces a way of combining the hashing results to provide an overall recall of 98.7%, a precision of 99.3% and an accuracy of 98.2% which are higher than the detection rates for the independent hashes. Since the technique uses static analysis, it is safe against malware that evade sandboxes and dynamic analysis environments. As a way of controlling the risk introduced by the human factor in security system, we present a quantitative measure for how malicious the file is. The limitation of this system is that one needs a starting baseline of the database of malicious file hashes. However, the use of a simple database working with light weight analysis scripts reduces the impact of this limitation. Each file is saved with its unique identifier of the MD5 so that there is no replication of file Hashes in the Database of Hashes. Each search carried out requires the return of the MD5 as the file identifier which keeps with the standard used by many

online analysis tools. Therefore, the method can also be open sourced to allow for customised manipulation and extension of the algorithms.

IX. Conclusion and Future Work

The 97.1% true positive detection rate with a trade-off of 0.296% false positive detection achieved in this study are very promising. Our system was designed using light weight tools which makes it efficient and fast. In malware detection, the objective is to build a filter like system and with this work, we introduce a way of detecting malicious PE type files without the need for dynamic analysis unless the result is inconclusive. This advantage and the results achieved provide a positive argument for the proposed technique. We are working on combining this technique with a heuristic detection method to create a new detection. Preliminary results already show promising results.

Acknowledgment

This work was carried out with the support of Nettitude Ltd and a special thank you to malshare.com that allowed us heavy malware downloads daily.

References

[1] "AV-TEST – The Independent IT-Security Institute." [Online]. Available: http://www.av-test.org/en/statistics/malware/. [Accessed: 14-Jul-2015].

[2] "McAfee Labs Threats Report," Nov-2015. [Online]. Available: http://www.mcafee.com/uk/resources/reports/rp-quarterly-threats-nov-2015.pdf. [Accessed: 01-Feb-2016].

[3] "Approaches to Integrated Malware Detection and Avoidance." [Online]. Available: https://www.rsbac.org/doc/media/nordse98.php. [Accessed: 16-Mar-2016].

[4] "Operating system market share." [Online]. Available: http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0. [Accessed: 25-Jul-2015].

[5] J. Pearl, "Decision making under uncertainty," ACM Comput. Surv., vol. 28, no. 1, pp. 89–92, Mar. 1996.

[6] Y. Li, S. C. Sundaramurthy, A. G. Bardas, X. Ou, D. Caragea, X. Hu, and J. Jang, "Experimental Study of Fuzzy Hashing in Malware Clustering Analysis," presented at the 8th Workshop on Cyber Security Experimentation and Test (CSET 15), 2015.

[7] R. Perdisci, A. Lanzi, and W. Lee, "Classification of packed executables for accurate computer virus detection," Pattern Recognit. Lett., vol. 29, no. 14, pp. 1941–1946, Oct. 2008.

[8] C. Oprisa, M. Checiches, and A. Nandrean, "Locality-sensitive hashing optimizations for fast malware clustering," in 2014 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2014, pp. 97–104.

[9] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," Digit. Investig., vol. 3, Supplement, pp. 91–97, Sep. 2006.

[10] "ModSecurity Advanced Topic of the Week: Detecting Malware with Fuzzy Hashing," Trustwave. [Online]. Available: https://www.trustwave.com/Resources/SpiderLabs-Blog/ModSecurity-Advanced-Topic-of-the-Week--Detecting-Malware-with-Fuzzy-Hashing/. [Accessed: 22-Jan-2016].

[11] Dunham Ken, "A fuzzy future in malware research," The ISSA J., vol. 11, no. 8, pp. 17–18, 2003.

[12] Georg Wicherski, "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, 2009, vol. 1–1.

[13] M. M. Gupta and J. Qi, "Fuzzy Logic and Uncertainty ModellingTheory of T-norms and fuzzy inference methods," Fuzzy Sets Syst., vol. 40, no. 3, pp. 431–450, Apr. 1991.

[14] L. A. Zadeh, "The role of fuzzy logic in the management of uncertainty in expert systems," Fuzzy Sets Syst., vol. 11, no. 1–3, pp. 199–227, 1983.

[15] B. Więckowski, "Review of Proof theory for fuzzy logics. Applied Logic Series, vol. 36," Bull. Symb. Log., vol. 16, no. 3, pp. 415–419, 2010.

[16] G. Shafer, A mathematical theory of evidence. Princeton, NJ: Princeton Univ. Press, 1976.

[17] B. G. Buchanan and E. H. Shortliffe, Eds., Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project. Reading, Mass: Addison-Wesley, 1984.

[18] D. E. Heckerman and E. H. Shortliffe, "From certainty factors to belief networks," Artif. Intell. Med., vol. 4, no. 1, pp. 35–52, Feb. 1992.

[19] DigitalNinja., "Using Fuzzy Hashing Techniques to Identify Malicious Code," Apr. 2007.

[20] David French and William Casey, "Fuzzy Hashing Techniques in Applied Malware Analysis," Results of SEI Line-Funded Exploratory New Starts Projects CMU/SEI-2012-TR-004, Aug. 2012.

[21] S. Arik, T. Huang, W. K. Lai, and Q. Liu, Neural Information Processing: 22nd International Conference, ICONIP 2015, Istanbul, Turkey, November 9-12, 2015, Proceedings. Springer, 2015.

[22] "Tracking Malware with Import Hashing," M-unition. [Online]. Available: https://www.mandiant.com/blog/tracking-malware-import-hashing/. [Accessed: 14-Jul-2015].

[23] A. Azab, R. Layton, M. Alazab, and J. Oliver, "Mining Malware to Detect Variants," in Cybercrime and Trustworthy Computing Conference (CTC), 2014 Fifth, 2014, pp. 44–53.

[24] A. P. Namanya, J. P. Disso, and I. U. Awan, "Evaluation of automated static analysis tools for malware detection in Portable Executable files," in 2015 31st UKPEW, University of Leeds, 2015, pp. 81–95.

[25] "radare." [Online]. Available: http://www.radare.org/r/. [Accessed: 23-Mar-2016].

[26] "guelfoweb/peframe," GitHub. [Online]. Available: https://github.com/guelfoweb/peframe. [Accessed: 03-Jul-2015].

[27] "erocarrera/pefile," GitHub. [Online]. Available: https://github.com/erocarrera/pefile. [Accessed: 01-Jun-2015].

[28] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," 2006, pp. 233–240.