2014 Fifth Cybercrime and Trustworthy Computing Conference

Mining Malware To Detect Variants

Ahmad Azab [a], Robert Layton [a], Mamoun Alazab [b], Jonathan Oliver [c]

[a]Internet Commerce Security Laboratory, Federation University Australia, VIC 3350, Australia
a.azab@icsl.com.au; r.layton@icsl.com.au

[b]College of Engineering and Technology, American University of the Middle East
Mamoun.Alazab@aum.edu.kw

[c]Trend Micro Australia
jon_oliver@trendmicro.com

***Abstract*—Cybercrimecontinues to be agrowing challenge and malware is one of the most serious security threats on the Internet todaywhich have been in existence from the very early days.Cyber criminals continue to develop and advance their malicious attacks. Unfortunately, existing techniques for detecting malware and analysing code samples are insufficient and have significant limitations. For example, most of malware detection studies focused only on detection and neglected the variants of the code. Investigating malwarevariants allows antivirus products and governments to more easily detect these new attacks, attribution, predict such or similar attacks in the future, and further analysis. The focus of this paper is performing similarity measures between different malware binaries for the same variant utilizing data mining concepts in conjunction with hashing algorithms. In this paper, we investigate and evaluate using the Trend Locality Sensitive Hashing (TLSH) algorithm to group binaries that belong to the same variant together, utilizing the k-NN algorithm. Two Zeus variants were tested, TSPY_ZBOT and MAL_ZBOT to address the effectiveness of the proposed approach.We compare TLSH to related hashing methods (SSDEEP, SDHASH and NILSIMSA)that are currently used for this purpose. Experimental evaluation demonstrates that our method can effectively detect variants of malware and resilient to common obfuscations used by cyber criminals.Our results show that TLSH and SDHASH provide the highest accuracy results in scoring an F-measure of 0.989 and 0.999 respectively.**

Keywords—Cybercrime, Cyber Security, Malware, Profiling, similarity, Hacking.

I. INTRODUCTION

Cybercrime causes significant damage each year, and has turned into a mature crime category [1-3]. The sophistication of targeted and organised crime has increased dramatically, using advanced techniques to perpetuate cybercrimes[4, 5]. The growth of the Internet has resulted in the increasing opportunities for computer attacks and intrusions [6]. Recent trends in binaries (code) designed for financial fraud purposes indicate their increasing complexity of software capability and they are also evolving rapidly as the Internet provides more opportunities for automated financial activities. As a result, the financial damage caused has dramatically increased in recent years [12][13][41].

The majority of the malicious software is recycled and hasn't been written from the beginning. In 2014 Symantec, stated in its reports [7] that the number of absolutely new malware families created reduced,as malware coders worked to perfect existing malware. In 2010 Symantec detected more than 286 million new malware variants [8]closely 90,000 unique variants of Zeus toolkit. More recently, over 20 million variants found in 2013 alone [9]. The similarity in many botnet families like in Hlux, Waledac, Nuwar,Kelihos and Storm is an evidence that all bots advanced by the same botnet crew, details in [10]. More recently, this year Fox-IT InTELLin their technical report [11] verified that the malware family Tilonlinked with Zeus and SpyEye malware family. Alazab in his recent article [39] argued that malware has distinct features from each other's which can be used for attribution and further analysis, also stated that lack of a comprehensive analysis of malware, not only for the purpose of detectingmalware, makes it harder to identify novel or existence attacks, characterize the basis of attacks, and predict such or similar attacks in the future.

Literature surveys on malware detection have shown that there is no single technique that could detect all types of malware, as most rely on syntactic properties and ignore the semantics of malicious code [12]. However, generally there are two techniques commonly used for malware detection, signature-based detection and anomaly-based detection [13-15] [42]. Anti-Virus engines use malware signatures to detect known malware. Amalware signature is a byte sequence that uniquely identifies a specific malware. Typically, a malware detector uses the malware signature to identify the malware like a fingerprint. Most AV engines are supplied with a database containing information of existing malware to identify maliciousness, by looking for code signatures such as byte

978-1-4799-8825-9/15 $31.00 © 2015 IEEE
DOI 10.1109/CTC.2014.11

sequences while scanning the system. A malware detector scans the system for characteristic byte sequences or signatures that match with the one in the database and declares the existence of malware blocking its access to the system. The signature matching process is called signature-based detection and most traditional AV engines use this method. It is a very efficient and effective method to detect known malware [16]. But, the major drawback is the inability to detect new or unknown malicious code. The signature generation involves manual processing and requires strict code analysis. To overcome signature based methods, polymorphic malware have an in-built polymorphic engine that can generate new variants each time it is executed and a new signature is generated. Therefore, signature based approaches fail to detect such malware. On the other hand, anomaly-based detection uses the knowledge of normal behaviour patterns to decide the maliciousness of a program code. This approach has the ability to detect some zero day attacks. However, it is very difficult to accurately specify the system or program's behaviour and thus these approaches usually are resulting in more false positives than signature based methods.

Malware coders are taking advantage of our increased reliance on digital systems, available digital resources, and increased connectivity and activity through Internet.Hence, identifying malware is one of the major concerns in information security. The focus of this paper is performing similarity measures between different malware binaries for the same variant utilizing data mining concepts in conjunction with hashing algorithms, specifically TLSH algorithm proposed by Trend Micro. These measurescan then be used to recognize future attacks, improve malware detection tools, develop better user education about the dangers of malicious software and ultimately to detect different new binaries of malware with low detection time (zero day detection). The latter reason is motivated that new variants of a malware family are released at an enormous rate compared to releasing new families, and also that the reliance of new malware families on old families has been proven.

This paper investigates using hashing algorithms for detecting when a piece of a malware belongs to an existing, known variant. This problem has been investigated in previous literature; our contribution is to compare the recently proposed TLSH algorithm with previously proposed approaches. In doing this, we also outline the role crime toolkits play in increasing the scale of malware by simply creating new variants of the same code. We evaluate the algorithms using the k-NN algorithm, comparing the F-measures scored on a labelled dataset.We find that TLSH is efficient ingrouping different binaries to the same variant.

The rest of the paper is organized as follows. Section II gives a brief overview about botnet and Zeus. Section III provides some related work for detecting malware. Section IV describes the different hashing algorithms used in this paper. Section V explains k-NN algorithm and F-measure equation. Section VI shows our analysis procedure and the results from our experiment for the provided datasets utilizing the different hashing algorithms. Section VII discusses the attained results for the aforementioned section and finally section VIII concludes this work.

II. CRIME KITS: OLD WINE IN A NEW BOTTLE

Crime ware toolkitsasexploit kits and attack kitsare serious threats where exploit kits are used bycybercriminals to distribute the malwarebinaries created by the attack kits.There is also a risk of escalation if the tools become more secure from detection.Cybercrime is enabled by a number of factors, including technical, policy and regulatory issues[17].

Today the goals of those releasing binaries have moved from showing off skills in coding and fun, to financial gain[18].Binaries attacks have become more organized and purposefully directed. Botnets in particular is a clear example of this trend. Botnets are literally vast numbers or armies of remotelycontrolled computers, or 'zombies'. These computers are compromised and then infected with software robots, or bots, that allow the zombie computers to be controlled remotely through established command and control channels (C&C). Collectively, under the control of C&C servers, botnets can become powerful and effective slave computing assets that can be rented for illegal activities. Such activities include phishing attacks, installing backdoors or rootkits on host systems to obtain private information, sending spam for advertising, and launching large scale DistributedDenial-of-Service(DDoS) attacks.

Since ZeuS first appearance in 2007, it has grown into one of the most dominate families to steal banking and private credentials. Many versions of ZeuS have been thoroughly investigated by the security community because of it prevalence. In May of 2011 the full source code of the infamous crimeware toolkit, ZeuS, had been leaked onto various Internet sites. This has led to the development of several centralized trojans based on ZeuS, such as ICE IX, KINS, and the more successful Citadel. Also, decentralized P2P Trojans based on ZeuS appeared in September 2011 as P2P ZeuS or *GameOverZeuS*. It is a significant improvement from all other versions of ZeuSbecause it replaces the centralized server with a P2P network[19].

The ZeuS Kit is a highly successful, easy to use suite of toolsthat allows relatively unskilled criminals to create and manage a botnet capable of stealing a wide variety of information from victims' machines. The real power of ZeuS comes from its use of zero day malware, 'Man in the Browser' techniques and its advanced web injection engine. All login credentials entered through the browser can be stolen, and through creative use of web injection the victim can be coerced into giving away far more information than they would ordinarily submit.

III. RELATED WORK

In recent years machine learning and data mining have been the focus of many malware researchers and analysts to counter the challenge of obfuscation techniques and malware detection[20]. Data mining is also referred to as knowledge discovery in databases. Frawley[21]define it as "*The nontrivial extraction of implicit, previously unknown, and potentially useful information from data*". It is also defined as "*The*

science of extracting useful information from large data sets or databases"[22]. To our knowledge, Schultz et al. [23] in 2001 were the first to apply datamining to the detection of different malicious programs based on their respective binary codes (program headers, strings and byte sequence) on several classifiers. Few years later, in 2004, Kolter et al.[24] improved the results byusing n-grams of byte codes as features, and applied several learning methods.

In 2004, Sung et al.in [25]and[26]proposed a method for computing the similarity between executablesof API calling sequences made in an attempt to detect polymorphic and metamorphic malwares. They defined signatures as an API sequence of calls and started the reverse engineering process from decompressed 16 binaries, which are then passed through a PE file parser, then extracted and mapped the sequence of Windows API calls, and lastly passed them through the similarity measure module, where similarity measures such as, Euclidian distance, Sequence alignment, Cosine measure, extended Jaccard measure, and the Pearson correlation measure were used.Although these similarity measures enable SAVE to detect polymorphic and metamorphic malwares efficiently against 8 malware scanners, the main drawbackis not considering the frequency appearance of the calls, which would add another detection layer to the overall system.In 2005, David and Michael [27] added a temporal consistency element to the system call frequency and calculated the frequency of API system call.In 2006, Kolter[28]described the use of machine learning and data mining to detect and classify malicious executables. Kolter tested several classifiers includingIBk, Naïve Bayes, support vector machines (SVMs), decision trees, boosted NaïveBayes, boosted SVMs, and boosted decision trees. Kolter found that SVMperformed exceptionally well and fast as compared to the other classifiers. However, this work did not focus on measuring the malware similarity.

In 2010, Shankarapani et. al[29] showed that the frequency of Windows API can be used to classify and detect malware with good accuracy. Authors have performed a static analysis to measure the similarity for 1,593 executables, of malware and benign. Two analysis methods have been used based on the frequency of occurrence of each Windows API. First, similarity analysis used to compute the mean value for 3 similarity measures (Cosine measure, extended Jaccard measure) have been used on the dataset. Second, SVM kernel RBF used to classify malware and benign dataset. However, the result of the Receiver Operating Characteristic (ROC) curve wasn't high rate compare to earlier studies.

IV. TLSH, NILSIMSA, SDHASH AND SSDEEP

This section explains the different hashing algorithms that are used to group malwarebinaries for the same variant together.

A. SSDEEP

SSDEEP[30]is described as a Context Triggered Piecewise Hash. It splits files into segments using context (a rolling hash) and identifies files as being similar if pieces and sequences of these pieces match.Given a file, F, SSDEEP generates the digest using a 3 step process:

1. Split the document into distinct segments using a rolling hash.
2. Generate a 6 bit value for each segment using a base65 encoding.
3. Generate the output digest by concatenating the values from step (2).

The similarity between two digests is determined by calculating the edit distance between the two digests using a dynamic programming approach. The output score is normalized to a range from 0 (no match) to 100 (identical, or a very close match). Typically digests with similarity scores >= 1 are considered "similar".

B. SDHASH

SDHASH[31, 32] adopts an approach closer to standard machine learning – it extracts features of significant length from files – and documents are identified as being similar if they share features. Given a file, F, SDHASH generates the digest using a 3 step process:

1. Identify 64 byte sequences which satisfy heuristic rules for their entropy.
2. Insert the sequences identified in step (1) into a series of Bloom filters.
3. Generate the output digest by encoding the series of Bloom filters from step (2).

The similarity between two digests is determined by calculating a normalized entropy measure between the two digests. SDHASH also normalizes the score to a range from 0 (no match) to 100 (identical, or a very close match). Typically digests with similarity scores >= 1 are considered "similar".

C. NILSIMSA

NILSIMSA[33]is a locality sensitive hashing algorithm focusing on using histograms to create the hash. It was designed originally to address the problem of detecting spam variants under small changes.

NILSIMSAcompares two inputs by computing histograms of trigrams. A sliding window of 5 characters moves along the input. Each time a new character is seen, each of the trigrams in the five character window is computed, and passed to a hashing algorithm h, such that h(i) is a value between 0 and 255. The number of times each value (between 0 and 255) is encountered is the histogram for that input. Next, the average of the buckets is computed and each bucket is assigned a 1 if it is above the average, or 0 if below, producing a 32 byte code.

These codes are then bitwise compared to compute the similarity score. The comparison score is the number of bits similar in the two codes, minus 128 (as this is the number of

bits likely to be the same in randomly generated codes). This gives a score between -128 and +127, with scores above 54 considered good matches in the original spam-based application. However in this application we simply used the inverse of this score as a distance metric.

D. TLSH

TLSH [34, 35] is a locality sensitive has closer in spirit to the NILSIMSAhash than the SSDEEP and SDHASH digests. Locality sensitive hashes extract a multitude of features from documents, and documents are identified as being similar if a critical mass of this multitude of features has similar profiles. Given a file, F, TLSH generates the digest using a 5 step process:

1. Use a sliding window to populate an array of bucket counts.
2. Calculate the quartile points of the bucket counts.
3. The digest header is a function of (i) the length of the file (ii) the quartile points calculated in step (2), and (iii) a checksum.
4. The digest body is generated by processing the bucket counts, turning each bucket count into a pair of bits in the range 0 to 3, based on bucket's value compared to the quartile points.
5. Generate the output digest by concatenating the digest header from step (3) and the digest body from step (4).

The distance between two digests is determined by summing the distance between the digest headers and the digest bodies. Two digest bodies have a distance which is the approximate Hamming distance between the two digest bodies. The digest headers include overall document information such as encoded approximate file length, and other global parameters describing the histogram of hash counts. The distance between two digest headers is determined as a function of the difference between header values. The resulting distance score between two digests ranges from 0 to 1000+. The recommended usage is that digests with a distance <= 100 are "similar", and that digests with a distance > 100 are "not similar". However, with TLSH there is a lot more flexibility with the threshold score than with SSDEEP and SDHASH.

V. K-NN

One of the main goals of machine learning is the ability to build computer systemsthat can adjust and learn from their experience. K-NN is a simple supervised machine learning algorithm that is used for classifying objects based on closest training instances in the feature space. It has been employed in many applications in data mining, statistical pattern recognition and many others. The object is classified based on a majority vote of its k nearest neighbors /low distance to the object. There are some measuring techniques that could be used to measure distance between the training object and the test object such as Bray-Curtis, Euclidean, correlation, Canberra, Manhattean, Chebyshev, Dice, Cosine, and Hamming distances.

In our experiments, the K-nearest neighbours are compute as follows with K:

1. Store all training samplesx_i^j in memory.
2. Determine the parameter K = number of nearest neighbors beforehand. In our experiment, k is chosen to be 5.
3. Measure the distance between the query-instance (x) and all the training samples x_i^j . (any distance algorithm can be used to) such as:

$$dis(x, x_i^j) \sqrt{\sum_{i=1}^{d} x(i) - x_i^j \ (i)^2} \tag{1}$$

4. Find the K-minimum distance between the query-instance (x) and each $Kj_{min}^1, j_{min}^2, \dots, j_{min}^k$.
5. Get all categories of training data for the sorted value under K.
6. Find the weighted distance of the query-instance (x) from each of the k nearest points as follows:

$$w = 1 - \frac{dis(x, x_i^j)}{\sum_{i=0}^{k} dis(x, x_i^j)} \tag{2}$$

As our performance measure, we perform a standard grid search with 10-fold cross validation to determine the best parameters for each classifier. We measure the performance of the classifier using the performance metricF-measurealso known as F-score, which is a measure of a test's accuracyfor binary classification functions that is based on the harmonic mean for the classifier's precisionand recall.The precision is the probability of records classified as positive which are classified correctly.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

The Recall is the probability of positive records that have been correctly identified.

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

F-measure is a measure of a test's accuracy by combining recall and precision scores into a single measure of performance, usually it is between 0.0 and 1.0, closer to 1 being a good score and closer to 0.0 being a poor score. Therefore it can be interpreted as a measure of overlapping betweenthe true and estimated classes (other instances, i.e. TN, are ignored), ranging from (no overlap at all to (complete overlap).

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5}$$

F-measure is one of the most accurate metrics for evaluation, especially for unbalanced datasets [36].

VI. EXPERIMENTALEVALUATION

This section provides first a description for the used dataset in our experiment. Then, it shows the results in applying TLSH hashing algorithm to group binaries for the same variant and differentiate binaries for different variants using the defined thresholds by Trend Micro. After that, the effectiveness in using k-NN for grouping binaries for the same variant or differentiating binaries for different variants is illustrated.

Finally, the efficiency in using the other hashing algorithm as SSDEEP, SDHASH and NILSIMSA with k-NN for classifying the binaries as being from the same or different variants is demonstrated.

A. Dataset

Two Zeus datasets were provided by Trend Micro to evaluate the use of TLSH and compare it with the other hashing algorithms. The two datasets hold binaries for two different variants, TSPY-ZBOT and MAL_ZBOT. The first dataset, TSPY_ZBOT, contains 856 binaries where the second dataset, MAL_ZBOT, contains 22 binaries.

Each binary has a unique SHA256 value, which indicates that the binaries are not redundant.

B. TLSH Results

TLSH was tested, using all the possible parameters combinations, in terms of grouping malware binaries for the same variant together and differentiating binaries for different variants. Two binaries belong to the same variant if the hash difference value is less than a defined threshold; otherwise they belong to two different variants. The threshold is given in the TLSH walkthrough guide. The threshold is defined as 200 if using 128 buckets for hashing or as 400 if using 256 buckets for hashing.

1) Grouping Binaries for The Same Variant

For the first experiment, we used the defined thresholds in order to group binaries for the same variant. Table I below shows the results that were attained by applying 128 buckets_1Byte checksum hash, 128 buckets_3Bytes checksum hash, 256 buckets_1Byte checksum hash and 256 buckets_3Bytes checksum hash using both "tlsh.diff" and "tlsh.xdiff" functions. The differences were computed between each binary with all other binaries within the same variant, with the note that the difference with the file itself was not included as well as the redundant differences between two binaries.

The accuracy result is defined as the number of the differences below the threshold over the total number of the differences.For each hash parameter, the average distances were computed by summing all the distances in each variant and divide them over the total number of the differences for that variant. Column "average distance" shows that the chosen thresholds for the distances were not accurate which lead in having low accuracy results.

The first notice that was inferred from the table, using either 1 or 3 Bytes checksum did not affect the accuracy results, but slightly affected the average distances. The second notice, the use of 256 buckets provided better accuracy results compared to the use of 128 buckets. This is explained as the follow. The lowest average distance using 128 buckets was 211.53 for Mal variant and 212.82 for TSPY variant, which are above the defined threshold. However, the lowest average distance using 256 buckets was 398.88 for Mal variant and 366.21 for TSPY variant, which are below the defined threshold.

Table I. TLSH Variant Grouping Distances and Accuracy

TLSH				
Hash Parameters	***Variant***	***Function***	***Accuracy Result***	***Average Distance***
128 Buckets, 1 Byte Check Sum	TSPY	Length	0.230	304.887
128 Buckets, 1 Byte Check Sum	Mal	Length	0.100	356.268
128 Buckets, 1 Byte Check Sum	TSPY	Xlength	0.323	212.817
128 Buckets, 1 Byte Check Sum	Mal	Xlength	0.463	211.528
128 Buckets, 3 Byte Check Sum	TSPY	Length	0.23	304.891
128 Buckets, 3 Byte Check Sum	Mal	Length	0.100	356.273
128 Buckets, 3 Byte Check Sum	TSPY	Xlength	0.323	212.821
128 Buckets, 3 Byte Check Sum	Mal	Xlength	0.463	211.532
256 Buckets, 1 Byte Check Sum	TSPY	Length	0.244	458.288
256 Buckets, 1 Byte Check Sum	Mal	Length	0.134	543.623
256 Buckets, 1 Byte Check Sum	TSPY	Xlength	0.380	366.218
256 Buckets, 1 Byte Check Sum	Mal	Xlength	0.489	398.883
256 Buckets, 3 Byte Check Sum	TSPY	Length	0.244	458.292
256 Buckets, 3 Byte Check Sum	Mal	Length	0.134	543.628
256 Buckets, 3 Byte Check Sum	TSPY	Xlength	0.380	366.222
256 Buckets, 3 Byte Check Sum	Mal	Xlength	0.489	398.887

The final notice, "xdiff" parameter provided higher accuracy results compared to "diff" parameter,as "xdiff" provided distances that are close to be below the threshold more compared to the use of "diff" parameters. This is clear as the lowest distances were provided by applying the "xdiff" parameter either on 128 buckets or 256 buckets. The best accuracy results were attained by applying both "xdiff" and 256 buckets parameters achieving 0.380 and 0.489 for TSPY and Malvariants respectively.

2) Differentiating Binaries from Different Variants

The second experiment was conducted in order to check if the defined thresholdsare feasible in differentiating binaries that belong to different variants. Distance above or equal the threshold means that the binaries belong to different variants. Table IIbelow shows accuracy results in computing the distances between binaries in variant "Mal" with binaries in variant "TSPY", then dividing the number of distances over or equal the threshold over the total number of distances.

As the previous experiment, the use of 1 or 3 Bytes checksum did not affect the accuracy results but slightly affects the average distance.

Unlike the previous results, this table shows that using 128 buckets provided better accuracy results compared to using 256 buckets. Also, for each use of "xdiff" parameter, it provided lower accuracy results compared to the "diff" parameter. The reason of that that the "xdiff" parameter provided lower average distance compared to "diff" parameter to be closer to the threshold, thus identifying the samples as the same variant.

The best accuracy results were attained by both using 128 buckets and using "diff" parameters, providing 0.980.

Table II. Differentiating Binaries from Different Variants using TLSH

TLSH			
Hash Parameters	***Function***	***Accuracy Result***	***Average Distance***
128 Buckets, 1 Byte Check Sum	Length	0.980	421.857
128 Buckets, 1 Byte Check Sum	XLength	0.820	253.877
128 Buckets, 3 Byte Check Sum	length	0.980	421.862
128 Buckets, 3 Byte Check Sum	Xlength	0.820	253.881
256 Buckets, 1 Byte Check Sum	Length	0.963	614.654
256 Buckets, 1 Byte Check Sum	XLength	0.762	446.674
256 Buckets, 3 Byte Check Sum	length	0.963	614.659
256 Buckets, 3 Byte Check Sum	Xlength	0.762	446.678

3) Using k-NN with TLSH

For the third experiment, a classifier using k-NNmachine learning algorithm was built in order to address its performance in grouping binaries that belong to the same variant together. First, a classifier was trained using binaries from the two different variants. Then for classifying an untrained binary, first the distance is calculated with all instances used in the training phase. Then, the closest distances with 5 binaries are investigated, where the classified binary belongs to a certain variant if three out of the five binaries belong to that variant. In order to evaluate this approach, we applied cross validation using n=10 for the given dataset. Table IIIbelow illustrates the results for this experiment.

Table III. Grouping Binaries Using k-NN and TLSH

TLSH		
Hash Parameters	**Difference**	**F-measure results**
128 Buckets, 1 Byte Check Sum	Length	0.989
128 Buckets, 1 Byte Check Sum	XLength	0.987
128 Buckets, 3 Byte Check Sum	length	0.989
128 Buckets, 3 Byte Check Sum	Xlength	0.987
256 Buckets, 1 Byte Check Sum	Length	0.989
256 Buckets, 1 Byte Check Sum	XLength	0.989
256 Buckets, 3 Byte Check Sum	length	0.989
256 Buckets, 3 Byte Check Sum	Xlength	0.989

The table reflects that using k-NN classifier to group binaries together for the same variants outperforms the usage of a specific threshold, attaining 0.989. The main reason for that is logic since as illustrated earlier, binaries for different variants would have larger distances compared with binaries within the same variant. F-measure metric was used to evaluate the classifier since we have unbalanced classes instances.

It also can be inferred from the table that using either 1 or 3 Bytes checksum provided the same results. For 128 buckets, the "diff" parameter provided higher results compared to the use of "xdiff" parameters. However, both parameters provided the same results using 256 buckets.

C. SSDEEP Results

The second used algorithm was SSDEEP. Since no threshold was provided to group or differentiate binaries for the same or different variants, k-NN experiment was just conducted. Before that, Table IVbelow shows information about the average distances for the same variant as well as for different variants, by applying SSDEEP. It is to be noted that SSDEEP uses range 0-100 for the distances where 0 means non-similar hashes and 100 means similar hashes. For our experiment convenience, we used the below formula to have 0 as similar hashes and 1 as non-similar hashes.

$$Distance = 1 - \left(\frac{Distnace\ Result}{Maximum\ Result}\right) \quad (6)$$

Where Distance Result is the value attained by applying SSDEEP for two different binaries and Maximum Distance is 100.

It can be noted that SSDEEP scores a higher average distance results for different binaries variants (Mal-TSPY) compared to the same binaries variant average distance. Also, it is observed that the average distance for the same variant is high and close to 1, which indicates that SSDEEP barely matches binaries for the same version.

Table IV. SSDEEP Average Distances Results

SSDEEP		
Hash Parameters	***Variant***	***Average Distance***
SSDEEP	Mal	0.983
SSDEEP	TSPY	0.882
SSDEEP	Mal-TSPY	1.000

For the next experiment, SSDEEP was used in conjunction with k-NN in order to group binaries for the same variant. As before, the closest distances with 5 binaries are investigated in where the classified binary belongs to a certain variant if three out of the five binaries belong to that variant. Table Vbelow shows the result of applying k-NN with SSDEEP.

Table V.k-NN Results for Grouping the Same Variant Binaries in Conjunction with SSDEEP

SSDEEP	
Hash Parameters	***F-measure results***
SSDEEP	0.987

It is observed that SSDEEP scores high results in grouping binaries for the same variant using k-NN, attaining 0.987.

D. SDHASH Results

The third used algorithm tested was SDHASH. Again, since no threshold was provided to group or differentiate binaries for the same or different variants, k-NN experiment was just conducted. Before that, Table VIbelow shows information about the average distances for the same variant as well as for different variants. As SSDEEP, SDHASH uses range 0-100 for the distances where 0 means non-similar hashes and 100 means similar hashes. Equation (6) is used to have 0 as similar and 1 as non-similar.

Table VI. SDHASH Average Distances Results

SDHASH		
Hash Parameters	***Variant***	***Average Distance***
SDHASH	Mal	0.964
SDHASH	TSPY	0.876
SDHASH	Mal-TSPY	0.999

Almost the same as SSDEEP results, SDHASH scoreshigher average distance results for different binaries variants (Mal-TSPY) compared to the same binaries variant average distance. Also, it is observed that the average distance for the same variant is high and close to 1, which indicates that SDHASH barely matches binaries for the same version.

For the next experiment, SDHASH was used in conjunction with k-NN in order to group binaries for the same variant. As before, the closest distances with 5 binaries are investigated in where the classified binary belongs to a certain variant if three out of the five binaries belong to that variant. Table VIIbelow shows the result of applying k-NN with SDHASH.

Table VII. k-NN Results for Grouping the Same Variant Binaries in Conjunction with SDHASH

SDHASH	
Hash Parameters	***F-measure results***
SDHASH	0.990

It is shown that SDHASH scores high results in grouping binaries for the same variant using k-NN, attaining 0.990.

E. NILSIMSA Results

The final used algorithm was NILSIMSA. Since no threshold was provided to group or differentiate binaries for the same or different variants, k-NN experiment was just conducted. Before that, Table VIIIbelow shows information about the average distances for the same variant as well as for different variants. As both SSDEEP and SDHASH, NILSIMSA uses range 0-100 for the distances where 0 means non-similar hashes and 100 means similar hashes. Equation (6) is used to have 0 as similar and 1 as non-similar.

Table VIII. NILSIMSA Average Distances Results

NILSIMSA		
Hash Parameters	***Variant***	***Average Distance***
NILSIMSA	Mal	0.155
NILSIMSA	TSPY	0.259
NILSIMSA	Mal-TSPY	0.169

Unlike the aforementioned results, NILSIMSA scored close to 0 results for binaries in the same variant. However, it is noticed that the distance average for binaries in different variants is close to 0 as well and is lower compared to TSPY average distance.

For the next experiment, NILSIMSA was used in conjunction with k-NN in order to group binaries for the same variant. As before, the closest distances with 5 binaries are

investigated in where the classified binary belongs to a certain variant if three out of the five binaries belong to that variant. Table IXbelow shows the result of applying k-NN with NILSIMSA.

Table IX. k-NN Results for Grouping the Same Variant Binaries in Conjunction with NILSIMSA

NILSIMSA	
Hash Parameters	***F-measure results***
NILSIMSA	0.987

The evaluation shows that our approach is highly effective in terms of response time and malware variant detection.NILSIMSA scores high results in grouping binaries for the same variant using k-NN, attaining 0.987.

VII. Discussion

The use of different hashing algorithms to group binaries for the same variant as well as differentiating binaries from different variants is feasible. We noticed that the use of TLSH different parameters provided good results in terms of having lower average distances for binaries in the same variant compared to binaries from different variants. The use of manual defined threshold didn't provide good accuracy results; on the other hand the use of k-NN showed that it is feasible to group binaries for the same version with high F-measure results.

For both SSDEEP and SDHASH, the average distances for the same variant were lower compared to the different variants. On the other hand, both of them scored average distance close to 1 (highly different) for binaries within the same variant.As TLSH, k-NN with these algorithms provided high F-measure results in grouping binaries within the same variant as well as differentiating binaries from different variants with the notice that SDHASH scored the highest.

NILSIMSA showed different behaviour compared to the rest by having higher average distance for binaries within the same variant (TSPY) compared to binaries from different variants. The explanation that k-NN provided good accuracy results in NILSIMSA is that when classifying any binary, if three distances within the same variant are lower compared to all distances from different variants, that binary will be classified as that same variant. In our experiment, when classifying a binary from TSPY variant, the distance between this binary and all binaries in both TSPY and Mal variants are computed. Three distances with TSPY binaries were lower compared to all the distances with Mal binaries and were enough to classify that binary as TSPY variant.

VIII. Conclusion

The malware (malicious software) landscape is persistently growing and a major threat. Current malware engines use a combination of signatures, and heuristics detection based methods. There are millions of malware codes propagated daily but there are only a low number of malware families. In this paper we investigated the use of locality sensitive hashing algorithms for malware variant classification. Our application scenario is that of an antivirus product; we wish to determine whether a previously unseen piece of malware is related to a variant we do know about.This problem had been studied in the literature previously. We compared the TLSH method with others, including SSDEEP, SDHASH and NILSIMSA.We used two real cases samples from Zeus variants, namely, TSPY_ZBOT and MAL_ZBOT to address the effectiveness of the proposed approach.

In order to have a deeper understanding of the behaviour of the results, different statistics should be extracted as the distribution of the distances. This helps to evaluate the algorithms from different perspective. Also, to address the robustness of this approach for different hashing algorithms, different datasets should be provided and tested. Experimental evaluation demonstrates that our method can effectively detect variants of malware and resilient to common obfuscations used by cyber criminals, our results shows that TLSH and SDHASH provide the highest accuracy results in scoring an F-measure of 0.989 and 0.999 respectively.

Our study onlyfocused on performing similarity measures between different malware binaries on obfuscated malware. Future plan is to perform some reverse engineering (de-obfuscated) on malware first, then apply different hashing algorithms on the de-obfuscated malware and compare results. Another future work is to compare the performance in applying cost sensitive algorithms to include different variants for the same version, as conducted [37]in for Skype application.

Reference

[1] R. Smith, P. Grabosky, and G. Urbas, *Cyber Criminals on Trial*: Cambridge University Press, 2004.

[2] P. Grabosky and R. G. Smith, *Crime in the Digital Age: Controlling Telecommunications and Cyberspace Illegalities*: Transaction Publishers, 1998.

[3] P. A. Watters, S. McCombie, R. Layton, and J. Pieprzyk, "Characterising and predicting cyber attacks using the Cyber Attacker Model Profile (CAMP)," *Journal of Money Laundering Control,* vol. 15, pp. 430-441, 2012.

[4] R. Broadhurst, P. Grabosky, M. Alazab, and S. Chon, "Organizations and Cyber crime: An Analysis of the Nature of Groups engaged in Cyber Crime," *International Journal of Cyber Criminology,* vol. 8, pp. 1 - 20, 2014.

[5] R. Mukhtar, A. Al-Nemrat, M. Alazab, S. Venkatraman, and H. Jahankhani, "Analysis of firewall log-based detection scenarios for evidence in

digital forensics," *Int. J. Electron. Secur. Digit. Forensic,* vol. 4, pp. 261-279, 2012.

[6] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Information Security Governance: The Art of Detecting Hidden Malware," in *IT Security Governance Innovations: Theory and Research*, ed: IGI Global, 2013, pp. 293-315.

[7] Symantec Corporation, "Internet Security Threat Report, Vol 19," April 2014.

[8] Symantec. (2011). *Symantec Internet Security Threat Report: Trends for 2010, Volume 16*. Available: https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HI-RES.pdf

[9] Dell. (2014). *Dell Network Security Threat Report 2013*. Available: http://www.sonicwall.com/app/projects/file_downloader/document_lib.php?t=WP&id=129

[10] P. Bureau, "Same Botnet, Same Guys, New Code," in *Virus Bulletin International Conference*, Barcelona, Spain, 2011, pp. 10 - 13.

[11] Fox-IT InTELL. (2014). *Tilon/SpyEye2 intelligence report* Available: http://foxitsecurity.files.wordpress.com/2014/02/spyeye2_tilon_20140225.pdf

[12] K. Alzarooni, "Malware Variant Detection," Doctor of Philosophy, Department of Computer Science, University College London, London, 2012.

[13] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*, Alexandria, Virginia, USA, 2008, pp. 51-62.

[14] G. Lawton, "Virus Wars: Fewer Attacks, New Threats," *IEEE Computer Society,* vol. 35, pp. 22 - 24, 2002.

[15] B. Birrer, R. Raines, R. Baldwin, M. Oxley, and S. Rogers, "Using Qualia and Hierarchical Models in Malware Detection," *Special Issue on Intrusion and Malware Detection: Journal of Information Assurance and Security,* vol. 4, 2009.

[16] S. Venkatraman, "Autonomic Context-Dependent Architecture for Malware Detection," presented at the e-Tech 2009, International Conference on e-Technology, Singapore, 2009.

[17] P. A. Watters, A. Herps, R. Layton, and S. McCombie, "ICANN or ICANT: Is WHOIS an Enabler of Cybercrime?," in *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*, 2013, pp. 44-49.

[18] R. Broadhurst and P. Grabosky, *Cyber-Crime: The Challenge in Asia*: Hong Kong University Press, 2005.

[19] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus," in *Malicious and Unwanted Software: "The Americas" (MALWARE), 2013 8th International Conference on*, Fajardo, PR, 2013, pp. 116 - 123.

[20] R. Layton, P. Watters, and R. Dazeley, "Unsupervised authorship analysis of phishing webpages," in *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, 2012, pp. 1104-1109.

[21] W. Frawley, G. Piatetsky-shapiro, and C. Matheus, "Knowledge discovery in databases: An overview," *Al Magazine,* vol. 13, pp. 213–228, 1992.

[22] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*: A Bradford Book, 2001.

[23] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *IEEE Symposium on Security and Privacy*, Oakland, CA, 2001, pp. 38-49.

[24] J. Kolter and M. Maloof, "Learning to Detect Malicious Executables in the Wild," in *The Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, NY, USA, 2004, pp. 470 – 478.

[25] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables (SAVE)," in *20th Annual Computer Security Applications Conference,* , Tucson, AZ, USA, 2004, pp. 326-334.

[26] J.-Y. Xu, A. H. Sung, P. Chavez, and S. Mukkamala, "Polymorphic Malicious Executable Scanner by API Sequence Analysis," presented at the Proceedings of the Fourth International Conference on Hybrid Intelligent Systems, 2004.

[27] D. J. Malan and M. D. Smith, "Host-based detection of worms through peer-to-peer cooperation," in *The ACM workshop on Rapid malcode*, Fairfax, VA, USA, 2005, pp. 72-80.

[28] J. Z. Kolter and M. A. Maloof, "Learning to Detect and Classify Malicious Executables in the Wild," *Journal of Machine Learning Research,* vol. 7, pp. 2721-2744, 2006.

[29] M. Shankarapani, K. Kancherla, S. Ramammoorthy, R. Movva, and S. Mukkamala, "Kernel machines for malware classification and similarity analysis," in *The 2010 International Joint Conference on Neural Networks*, Barcelona 2010, pp. 1-6.

[30] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital Investigation,* vol. 3, Supplement, pp. 91-97, 9// 2006.

[31] V. Roussev, "An evaluation of forensic similarity hashes," *Digital Investigation,* vol. 8, Supplement, pp. S34-S41, 8// 2011.

[32] V. Roussev, "Data Fingerprinting with Similarity Digests," in *Advances in Digital Forensics VI*. vol. 337, K.-P. Chow and S. Shenoi, Eds., ed: Springer Berlin Heidelberg, 2010, pp. 207-226.

[33] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "An Open Digest-based Technique for

Spam Detection," *ISCA PDCS,* vol. 2004, pp. 559-564, 2004.

[34] J. Oliver, C. Chun, and C. Yanggui, "TLSH -- A Locality Sensitive Hash," in *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*, 2013, pp. 7-13.

[35] trendmicro. (2014, 8 September). *TLSH source code* Available: https://github.com/trendmicro/tlsh

[36] P. Christen, "Evaluation of Matching Quality and Complexity," in *Data Matching*, ed: Springer Berlin Heidelberg, 2012, pp. 163-184.

[37] A. Azab, R. Layton, M. Alazab, and P. Watters, "Skype Traffic Classification Using Cost Sensitive Algorithms," in *2013 Fourth Cybercrime and Trustworthy Computing Workshop (CTC)*, 2013, pp. 14-21.