SARVAM: Search And RetrieVAl of Malware

Lakshmanan Nataraj
University of California, Santa Barbara
lakshmanan_nataraj@ece.ucsb.edu

Dhilung Kirat
University of California, Santa Barbara
dhilung@cs.ucsb.edu

B.S Manjunath
University of California, Santa Barbara
manj@ece.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
vigna@cs.ucsb.edu

ABSTRACT

We present SARVAM, a system for content-based Search And RetrieVAl of Malware. In contrast with traditional static or dynamic analysis, SARVAM uses malware binary content to find similar malware. Given a malware query, a fingerprint is first computed based on transformed image features [19], and similar malware items from the database are then returned using image matching metrics. The current SARVAM database holds approximately 4.3 million samples of malware and benign executables. The system is demonstrated using a desktop computer with Ubuntu OS, and takes approximately 3 seconds per query to find the top matching malware. SARVAM has been operational for the past 15 months during which we have received approximately 212,000 queries from users. In this paper, we describe the design and implementation of SARVAM and also discuss the nature and statistics of queries received.

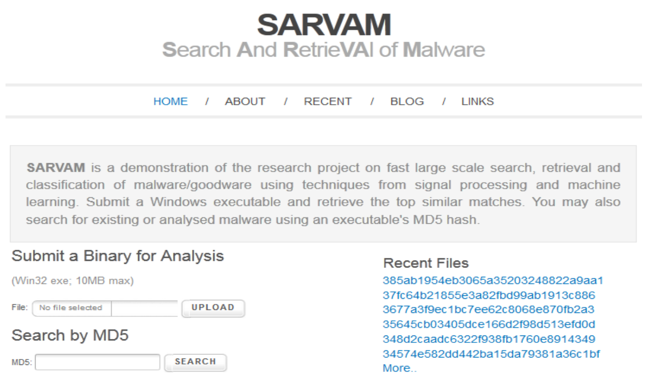
Keywords

Malware similarity, Content based search and retrieval, Malware images, Image similarity

1. INTRODUCTION

With the phenomenal increase in malware (on the order of hundreds of millions), standard techniques to analyze malware like static code analysis and dynamic analysis have become a huge computational overhead. Moreover, most of the new malware are only variants of already existing malware. Hence, there is a need for faster identification of these variants to catch up with the malware explosion. This in turn requires faster and compact signature extraction methods. For this, techniques from signal and image processing, data mining and machine learning that handle such large scale problems can play an effective role.

In this paper, we utilize signature extraction techniques from image processing and build a system, SARVAM, for large scale malware search and retrieval. Leveraging on past work in finding similar malware based on image similarity [17], we use these compact features for content-based search and retrieval of malware. These features are known to be robust, highly scalable and perform well in identifying similar images in a web-scale dataset of natural images (110 million) [7]. They are fast to compute and are shown to be 4000 times faster than dynamic analysis while having similar performance in malware classification [18] and also used in malware detection [12]. These image similarity features are computed on a large dataset of malware (more than 4 million samples) and stored in a database. For fast search and retrieval, we use a scalable *Balltree*-based Nearest Neighbor searching technique. This reduces the average query time to 3 seconds for a given query. We built SARVAM as a public web-based query system, (accessible at *http://sarvam.ece.ucsb.edu*), where users can upload queries and obtain similar matches for that query. The system has been active since May 2012 and we have received more than 212,000 samples since then. For a large portion of the uploaded samples, we were able to find variants in our database. Currently, there are only a few public systems that allow users to upload malware samples and obtain reports. To the best of our knowledge, SARVAM is the only system among them in finding similar malware. We briefly give an overview below.


SARVAM
Search And RetrieVAl of Malware

SARVAM is a demonstration of the research project on fast large scale search, retrieval and classification of malware/goodware using techniques from signal processing and machine learning. Submit a Windows executable and retrieve the top similar matches. You may also search for existing or analysed malware using an executable's MD5 hash.

Submit a Binary for Analysis
(Win32 exe; 10MB max)

File: No file selected UPLOAD

Search by MD5

MD5: SEARCH

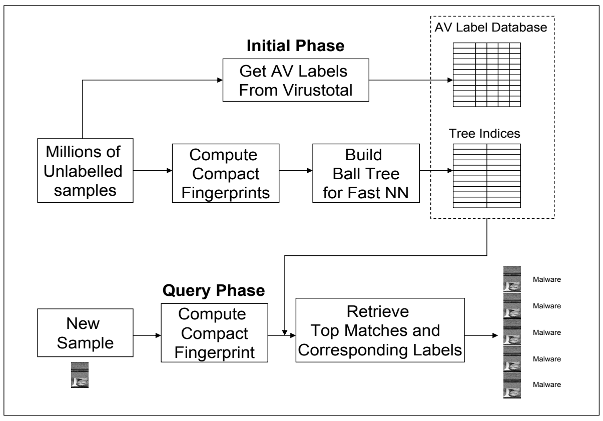
Recent Files
385ab1954eb3065a35203248822a9aa1
37fc64b21855e3a82fbd99ab1913c886
3677a3f9ec1bc7ee62c8068e870fb2a3
35645cb03405dce166d2f98d513efd0d
348d2caadc6322f938fb1760e8914349
34574e582dd442ba15da79381a36c1bf
More..

Figure 1: Web Interface of SARVAM

1.1 SARVAM Overview

A content-based search and retrieval system is one in which the content of a query object is used to find similar objects in a larger database. Such systems are common in the retrieval of multimedia objects such as images, audio and video. The objects are usually represented as compact descriptors or fingerprints based on the their content [24].

SARVAM uses image similarity fingerprints to compactly describe a malware. These effectively capture the visual (structural) similarity between malware variants and are used for search and retrieval. There are two phases in the system design as shown in Fig. 2. During the initial phase, we first obtain a large corpus of malware samples from various sources [1,3]. The compact fingerprints for all the samples in the corpus are then computed. To obtain similar malware, we use Nearest Neighbor (NN) method based on the shortest distance between the fingerprints. But the high dimensionality of the fingerprints makes the search slow. In order to perform Nearest Neighbor search quickly and efficiently, we construct a Balltree (explained in Sec. 2), which significantly reduces the search time. Simultaneously, we obtain the Antivirus (AV) labels for all the samples from Virustotal [4], a public service that maintains a database of AV labels. These labels act as a ground truth and are later used to describe the nature of a sample, i.e., how malicious or benign a sample is. During the query phase, the fingerprint for the new sample is computed and matched with the existing fingerprints in the database to retrieve the top matches. The various blocks of SARVAM are explained in the following sections.



Initial Phase
Get AV Labels From Virustotal
AV Label Database
Millions of Unlabelled samples
Compute Compact Fingerprints
Build Ball Tree for Fast NN
Tree Indices
Query Phase
New Sample
Compute Compact Fingerprint
Retrieve Top Matches and Corresponding Labels
Malware
Malware
Malware
Malware
Malware

Figure 2: Block schematic of SARVAM

The rest of the paper is organized as follows. In Sec. 2, the steps to compute the compact fingerprint from a malware and the fast Balltree-based Nearest Neighbor search method are explained. Sec. 3 explains the implementation details. The details on the uploaded samples are briefed in Sec. 4 while the limitations, related work and conclusion are mentioned in Sec. 5, Sec. 6 and Sec. 7 respectively.

2. COMPACT MALWARE FINGERPRINT

2.1 Feature Extraction

Our objective is to compute a robust and compact signature from an executable that can be used for efficient search and retrieval. For this, we consider techniques from signal and image processing where such compact signature extraction methods have been extensively studied. Our approach is based on a feature extraction technique as described in [17], which uses the GIST image features. The features are based on the texture and spatial layout of an image. These have been widely explored in image processing for contest-based image retrieval [16], scene classification [19, 28], and large scale image search [7]. The binary

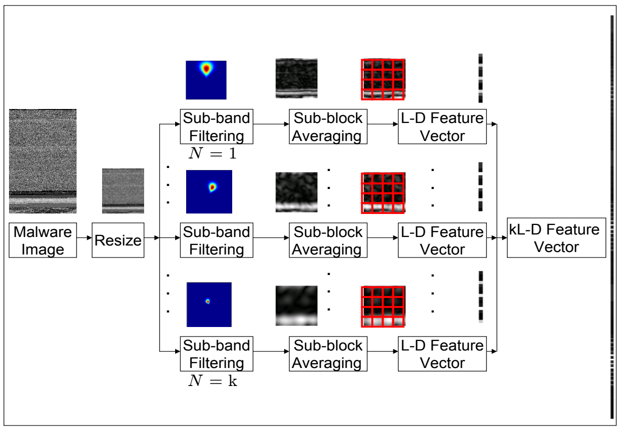


Figure 3: Block diagram to compute feature from a malware

content of the executable is first numerically represented as a discrete one dimensional signal by considering every byte value as an 8 bit number in the range 0-255. This signal is then "reshaped" to a two dimensional grayscale image. Let d be the width and h be the height of the "reshaped" image. While reshaping, we fix the width d and let the height h vary depending on the number of bytes in the binary. The horizontally adjacent pixels in the image correspond to the adjacent bytes in the binary and the vertically adjacent pixels correspond to the bytes spaced by a multiple of width d in the binary. The image is then passed through various filters that capture both the short-range and long-range correlations in the image. From these filtered images, localized statistics are obtained by dividing the filtered images into non-overlapping sub-blocks, and then computing the average value on those blocks. This is called sub-block averaging and the averages computed from all the filters are concatenated to form the compact signature. In practice, the features are usually computed on a smaller "resized" version of the image. This is done for faster computation and usually does not affect the performance. Feature computation details are given below.

Let $I(x, y)$ be the image on which the descriptor is to be computed. The GIST descriptor is computed by filtering this image through a filter bank of Gabor filters. These filters are band pass filters whose responses are Gaussian functions modulated with a complex sinusoid. The filter response $t(x, y)$ and its Fourier transform $T(u, v)$ are defined as:

$$t(x,y) = \frac{1}{(2\pi\sigma_x\sigma_y)} exp[-\frac{1}{2}(\frac{x^2}{{\sigma_x}^2} + \frac{y^2}{{\sigma_y}^2}) + 2\pi jWx] \quad (1)$$

$$T(u,v) = exp[-\frac{1}{2}(\frac{(u-W)^2}{(\sigma_u)^2} + \frac{v^2}{(\sigma_v)^2})] \quad (2)$$

where $\sigma_u = 1/2\pi\sigma_x$ and $\sigma_v = 1/2\pi\sigma_y$. Here, σ_x and σ_y are the standard deviations of the Gaussian functions along the x direction and y direction. These parameters determine the bandwidth of the filter and W is the modulation frequency. (x, y) and (u, v) are the spatial and frequency domain coordinates.

We create a filter bank by *rotating* (orientation) and *scaling* (dilation) the basic filter response function $t(x, y)$, resulting in a set of self-similar filters. Let S be the number of scales and O be the number of orientations per scale in a *multiresolution decomposition* of an image. An image is filtered using k such filters to obtain k filtered images as shown in Fig. 3. We choose $k = 20$ filters with 3 scales ($S = 3$), out of which first the two scales have 8 orientations ($O = 8$) and the last one has 4 ($O = 4$). Our experiments showed that having more scales or orientations did not improve the performance. Each filtered image is further divided into $B \times B$ sub-blocks and the average value of a sub-block is computed and stored as a vector of length $L = B^2$. This way, k vectors of length L are computed per image. These vectors are then concatenated to form a kL-dim feature vector called GIST. In SARVAM, we choose $B = 4$ to obtain a 320-dim feature vector. While computing the GIST descriptor, it is a common pre-processing step to resize the image to a square image of dimensions $s \times s$. In our experiments, we choose $s = 64$. We observed that choosing a value of s less than $s = 64$ did not result in a robust signature. Larger value of s increased the computational complexity, however, because of the sub-band averaging, this did not effectively strengthen the signature.

2.2 Illustration on Malware Variants

Here, we consider two malware variants belonging to Backdoor.Win32.Poison family. The grayscale visualizations of the variants are shown in Fig. 4. We can see that these two variants have small variations in their code. The difference image on the right shows that most parts in the difference are zero (shown as white). We compute features for these variants and then overlay the absolute difference of the first 16 coefficients of these features on the difference image (Fig. 4). One can see that there is a difference in features only in sub-blocks which also have a difference (shown in red in Fig. 4). Although only the difference is shown for one filtered image, this pattern holds for all other filtered images as well.

2.3 Feature Matching

Consider a dataset of M samples: $\{Q_i\}_{i=1}^{M}$, where Q_i denotes a sample. We extract a feature vector $G = f(Q)$, where $f(.)$ is the feature extraction function s.t.

$$Sim(Q_i, Q_j) \to Dist(G_i, G_j) < \delta \qquad (3)$$

where $Sim(Q_i, Q_j)$ represents the similarity between samples Q_i and Q_j, $Dist(.)$ is the distance function, and δ is a pre-defined threshold. Given a malware query, SARVAM first computes its image feature descriptor as explained above, and then searches the database for other feature vectors that are close to the query feature in the descriptor space. Straight forward way of doing this is to perform a brute-force search on the entire database, which is time consuming. Hence, we use an approximate Nearest Neighbor searching algorithm, which we explain in the next section.

2.4 Fast Nearest Neighbor Search

The dimensionality of the GIST feature vector is 320. For efficient nearest-neighbor search in high dimensional space, we use *Balltree* data structures [20]. A Ball, in n-dim Euclidean space R^n, is defined as a region bounded by a hyper sphere. It is represented as $B = \{\underline{c}, r\}$, where $\underline{c}$ is an n-dim vector specifying the coordinates of the ball's centroid, and r is the radius of the ball. A Balltree is a binary tree where each node is associated with a ball. Each ball is a minimal ball that contains all balls associated with its children nodes. The data is recursively partitioned into nodes defined by the centroid and the radius of the ball. Each point in the node lies within this region. As an illustration, Fig. 6 shows a binary tree, and a Balltree over four balls (1,2,3,4). Search is carried out by finding the minimal ball that completely contains all its children. This ball also overlaps the least with other balls in the tree. For a dataset of M samples and dimensionality N, the query time grows approximately as $O[N \log(M)]$ (as opposed to $O[NM]$ for a brute force search). We conduct a small experiment to compare the query time and build time. We choose 500 pseudorandom vectors of dimension 320. These are sent as queries to a larger pseudorandom feature matrix of varying sample sizes (from 100,000 to 2 Million) and same dimension. The total build time and total query time are computed for the cases of brute force search and Balltree-based search (Fig. 5). We see that there is a significant difference in the query time between the Balltree-based search and brute force search as the number of samples in the feature matrix increases. In the case of build time, the time taken to build a Balltree increases as the sample size increases. In practical systems, however, the query time is given more priority than the build time.

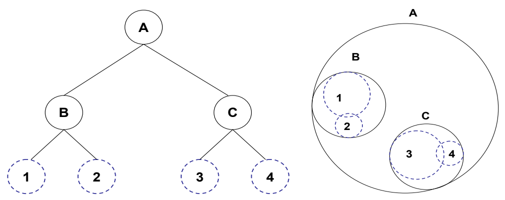


Figure 6: Illustration of: (a) Binary tree (b) Corresponding Balltree

3. SYSTEM IMPLEMENTATION

SARVAM is implemented on a desktop computer (DELL Studio XPS 9100 Intel Core i7-930 processor with 8MB L2 Cache, 2.80GHz and a 20 GB RAM) running on Ubuntu 10. The web server is built using Ruby on Rails framework with MySQL backend. Python is used for feature computation and matching. A MySQL database stores information about all the samples such as MD5 hash, file size and number of Antivirus (AV) labels. When a new sample is uploaded, its MD5 hash is updated in the database. All the uploaded samples are stored on disk and saved by their MD5 hash name. A Python script (daemon) checks the database for unprocessed keys and when it finds one, it takes the corresponding MD5 hash and computes the image fingerprint from the stored sample. Then, the top matches for that query are found and the database is updated with their MD5 hashes. A Ruby on Rails script then checks the database and displays the top matches for that sample. The average time taken for all the above steps is approximately 3 seconds.

3.1 Initial Corpus

The SARVAM database consists of approximately 4.3 million samples, most of which are malware. We also include a small set of benign samples from clean installations of various Windows OS. All the samples are uploaded to Virustotal to get Antivirus (AV) labels and these are stored in a MySQL database. Fig. 7 shows the distribution of the AV labels for all the samples in our initial corpus. As we can see, most samples have many AV labels associated with them, thus indicating they are most certainly malicious in nature. The corpus and the MySQL database are periodically updated as we get new samples. The AV labels of the

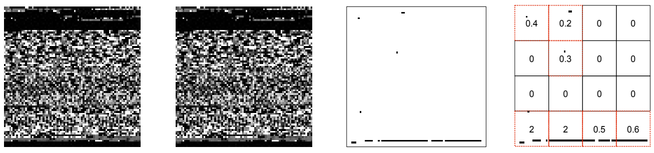


Figure 4: Grayscale visualizations of Backdoor.Win32.Poison malware variants (first two images) and their difference image (white color implies no difference). The last image shows the difference image divided into 16 sub-blocks and the absolute difference between the first 16 coefficients of the GIST feature vectors of the two variants are overlaid. The sub-blocks for which there is a difference are colored in red. We see that there is a variation in the feature values only in the red sub-blocks.

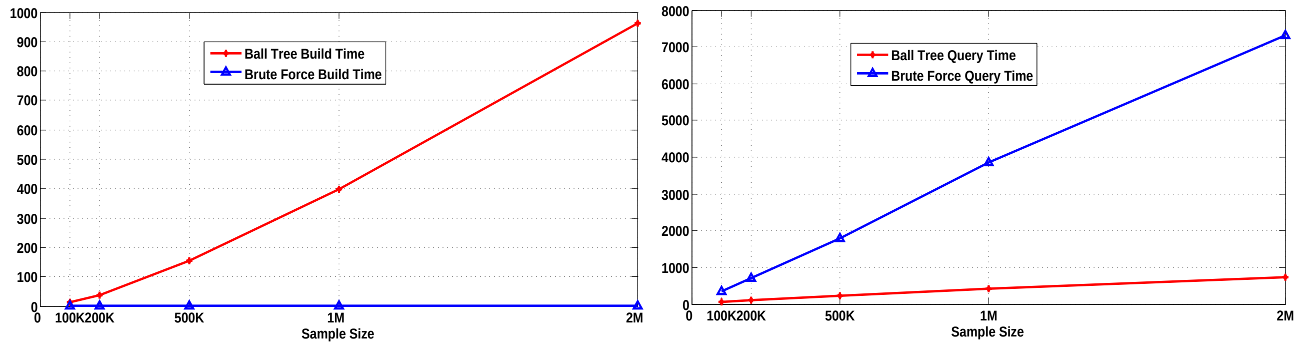


Figure 5: Comparison of Brute Force Search and Balltree Search: (a)Build Time (b)Query Time

samples are also periodically checked with Virustotal and updated if there are changes in the labels. This is because AV vendors sometimes take a while to catch up with the malware and hence, the AV labels may change.

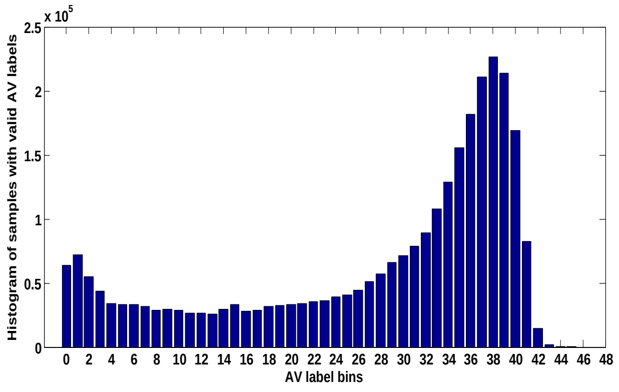


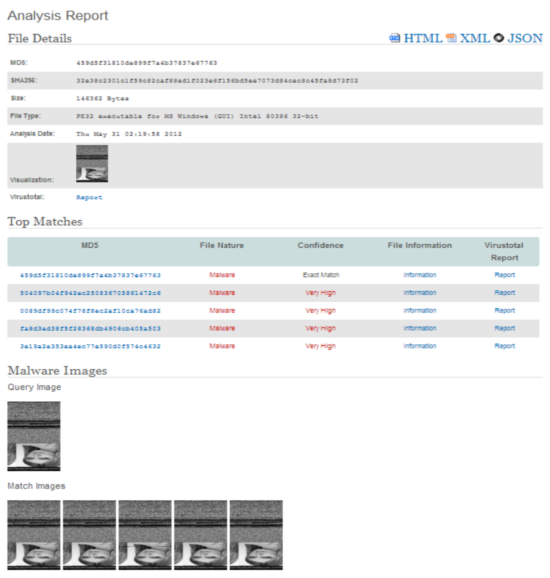
Figure 7: Distribution of the number of AV labels in the corpus

3.2 Web Interface

SARVAM has a simple web interface built on Ruby on Rails as shown earlier in Fig. 1. Some of the basic functionalities are explained below.

Search by upload or MD5 Hash: SARVAM currently supports two ways of search. In the first case, users can upload executables (maximum size 10 MB) and obtain the top matches. In the second, users can search for an MD5 hash and if the hash is found in our database, the top matches are computed. Currently, only Win32 excutables are supported but our method can be easily generalized to include a larger category of data.

Sample Reports: A sample query report is shown in Fig. 8. SARVAM supports HTML, XML and JSON versions. While HTML reports aid in visual analysis, XML and JSON reports can be used for script-based analysis.



Analysis Report

File Details HTML XML JSON

MD5:	459d5f31810de899f7a4b37837e67763
SHA256:	32e38c2301c1f59c82caf88ed1f023e6f156bd5ee7073d84cec8c45fa8d73f02
Size:	146362 Bytes
File Type:	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
Analysis Date:	Thu May 31 02:19:58 2012
Visualization:	
Virustotal:	Report

Top Matches

MD5	File Nature	Confidence	File Information	Virustotal Report
459d5f31810de899f7a4b37837e67763	Malware	Exact Match	Information	Report
504097b04f942ec250836705861472c6	Malware	Very High	Information	Report
0089df99c074f78f8ec2af10ca76ad82	Malware	Very High	Information	Report
fa8d3ed38f5f28368db4906cb405a503	Malware	Very High	Information	Report
3e19a2e353ea4ec77e590d0f574c4632	Malware	Very High	Information	Report

Malware Images

Query Image

Match Images

Figure 8: Sample HTML report for a query

3.3 Design Experiments

For a given query input, the output is a set of matches which are ranked according to some criterion. In our case, the criterion is based on the distance between the query and its top match. We set various thresholds to the distance and give confidence levels to the matches.

3.3.1 *Training Dataset*

Two malware are said to be variants if they show similar behavior upon execution. Although some existing works try to quantify such malware behavior [6, 23], it is not very straightforward and can result in spurious matches. An alternative is to check if the samples have same AV labels. Many works including [6, 23] use AV labels to build the ground truth. We evaluate the match returned for a query based on the number of common AV labels. From our corpus of 4.3 million samples, we select samples for which most AV vendors have some label. In Virustotal, the AV vendor list for a particular sample usually varies between 42 and 45 and in some unique cases goes down to 5. In order to not skew our data, we select samples for which at least 35 (approximately 75% - 80%) AV vendors have valid labels (*None* labels excluded). This resulted in a pruned dataset of 1.4 million samples.

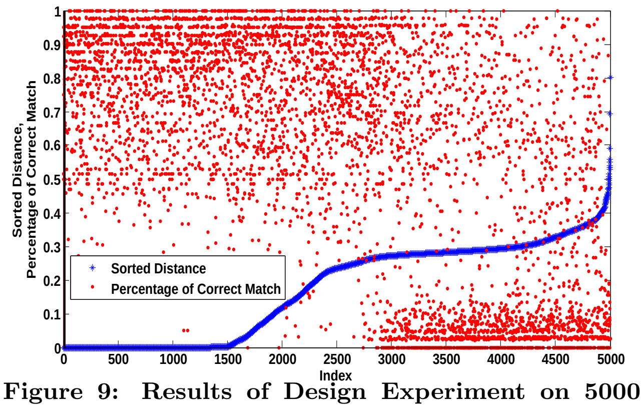


Figure 9: Results of Design Experiment on 5000 samples randomly chosen from the training set. Sorted Distance and corresponding percentage of correct match are overlaid on the same graph. A low distance value has a high match percentage while a high distance value has a low match percentage in most cases.

3.3.2 Validation

From the pruned dataset of 1.4 million samples, we randomly choose a reduced set R_s of length $N_{R_s} = 5000$ samples. The remaining samples in the pruned set are referred as the training set T_s. The samples from R_s are queries to the samples in T_s. First, the features for all the samples are computed. For every sample of R_s, the nearest neighbor among the samples of T_s is computed. Let $q_i = R_s(i)$, $1 \leq i \leq N_{R_s}$ be a query, m_i be its Nearest Neighbor (NN) match among the samples in T_s, d_i be the NN distance and AV_{sh} be a set of shared AV vendor keys (such as *Kaspersky, McAfee*). Both the query and the match will have corresponding AV labels (such as *Trojan.Spy, Backdoor.Agent*) for every shared vendor key. We are interested in finding how many matching labels are present between a query and its match, and its relation with the NN distance. The percentage of matching labels pm_i between a query and its match is defined as:

$$pm_i = \frac{\sum_{j=1}^{N_{AV_{sh}}} I(q_i[AV_{sh}(j)] = m_i[AV_{sh}(j)])}{N_{AV_{sh}}}, \; 1 \leq i \leq N_{R_s} \tag{4}$$

where $N_{AV_{sh}}$ is the total number of shared AV vendor keys, $q_i[AV_{sh}(j)]$ and $m_i[AV_{sh}(j)]$ are the AV labels of the query and its NN match for the i^{th} query and j^{th} AV vendor key and $I(.)$ is the Indicator function. We are interested in seeing which range of the NN distance d gives a high percentage of best AV label match pm. In order to visualize this, the distances are first sorted in ascending order. The sorted distances and the corresponding percentage of correct match are overlaid in Fig. 9. We observe that the percentage of the correct matches are highest for very low distances and they decrease as the distance increases. Our results were the same even we chose various random subsets of 5000 samples. Based on these results, we give qualitative tags and labels for the quantitative results as shown in Tab. 1.

3.4 Qualitative vs Quantitative Tags

For every query, we have the distance from its nearest neighbor and can compute the percentage of correct match between their labels. In reality, only the AV labels of the nearest neighbor are known and AV labels of the query may not be available. Hence, based on the NN distance and the number of AV labels that are present in a match, we give qualitative tags.

Intuitively, we would expect that a low distance would give the best match. A low distance means the match is very similar to the query and we give it a tag of *Very High Confidence*. As the distance increases, we give qualitative tags: *High Confidence*, *Low Confidence* and *Very Low Confidence* as shown in Tab. 1.

Very High Confidence Match: A Very High Confidence match usually means that the query and the match are more or less the same. They just differ in a few bytes. The example shown in Fig. 10 will help illustrate this better. The image in the left is the query image and the MD5 hash of the query is *459d5f31810de899f7a4b37837e67763*. We see an inverted image of a girl's face which is actually the icon of the executable. The image in the middle is the top match to the query with MD5 *fa8d3ed38f5f28368db4906cb405a503*. If we take a byte by byte difference between the query and the match, we see that most of the bytes in the difference image is zero. Only 323 bytes out of 146304 bytes (0.22%) are non-zero. The distance of the match from the query will usually be lesser than 0.1.

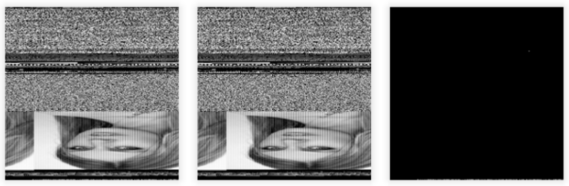


Figure 10: Example of a very high confidence match. The image in the left is of the query while in the middle image is of the top match. Shown in the right is the difference between the two. Only a few bytes in the difference image are non-zero.

High Confidence Match: When we talk about a high confidence match, most parts of the query and the top match are the same but there is a small portion that is different. In Fig. 11, we can see the image of the input query *1a24c1b2fa5d59eeef02bfc2c26f3753* in the left. The image of the top match *24faae39c38cfd823d56ba547fb368f7* in the middle appears visually similar to the query. But the difference image shows that 11,108 out of 80,128 non-zero values (13.86%). Most variants in this category are usually packed variants which have different decryption keys. The distance between the query and the top match will usually be between 0.1 and 0.25.

Low Confidence Match: For low confidence matches, a major portion of the query and the top match are different. We may not see any visual difference between the input query *271ae0323b9f4dd96ef7c2ed98b5d43e* and the top match *e0a51ad3e1b2f736dd936860b27df518* but the difference image clearly shows the huge difference in bytes (Fig. 12).

Table 1: Confidence of a Match

Distance d	Confidence Level	Percentage of pm	Median of pm	Mean of pm	Std. Deviation of pm
< 0.1	Very High	38.6	0.8462	0.7901	0.1782
(0.1,0.25]	High	15.24	0.7895	0.7492	0.2095
(0.25 ,0.4]	Low	44.46	0.1333	0.3454	0.3488
> 0.4	Very Low	1.7	0.0625	0.1184	0.1862

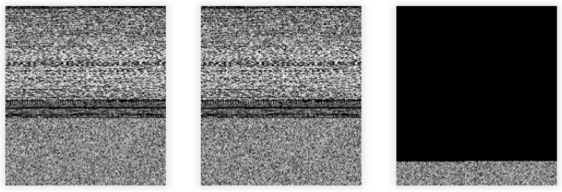


Figure 11: Example of a high confidence match. The image in the left is of the query while in the middle image is of the top match. Shown in the right is the difference between the two. A small portion in the difference image are non-zero.

These would usually be packed variants (UPX in this case). In the difference image, 75,353 out of 98304 non zero (76.6%). The distance is usually greater than 0.25 and less than 0.4. Low Confidence matches also end up in False Positives (meaning the top match may not be a variant of the query) and hence they are tagged as *Low Confidence*. In these cases, it is better to visually analyze the query and the top match before arriving at a conclusion.

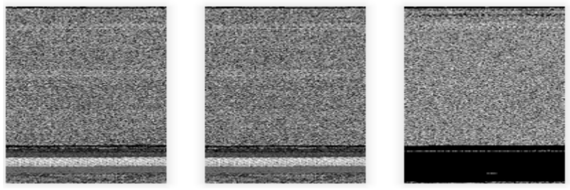


Figure 12: Example of a low confidence match. The image in the left is of the query while in the middle image is of the top match. Shown in the right is the difference between the two. A significant portion in the difference image are non-zero.

Table 2: Nature of a Match

No. of AV Labels	Qualitative Label
0	Benign
[1,10]	Possibly Benign
[11,25]	Possibly Malicious
[26,45]	Malicious
No data	Unknown

Very Low Confidence Match: For matches with Very Low confidence, in most of the cases the results don't really match the query. These are cases where the distance is greater than 0.4.

Apart from the confidence level, we also give qualitative tags to every sample in our database based on how many Antivirus (AV) labels it has. For this, we obtain the AV labels from Virustotal periodically. We use the count of the number of labels to give a qualitative tag for a sample as shown in Tab. 2.

4. RESULTS ON UPLOADED SAMPLES

SARVAM has been operational since May 2012. In this section, we detail the statistics of the samples that have been uploaded to our server, the results on the top matches and also the percentage of hits (with respect to AV labels) that our matches provide.

4.1 Statistics of Uploads

Distribution based on Month: From May 2012 onwards till Oct 2013, we received approximately 212,000 samples. In Fig. 13, we can see the distribution of the uploaded samples based on the uploaded month. We observe that most of the samples were submitted in Sep. 2012 and Oct. 2012 while the activity was very low in the months of Nov. 2012, Feb. 2013, Mar. 2013 and May 2013.

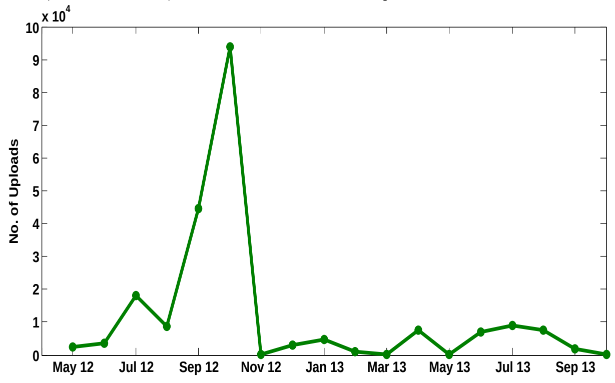


Figure 13: Month of Upload

Year of First Seen: In Fig. 14, we see the distribution of the year in which the samples were first seen in the wild by Virustotal. Most samples that we received are from 2011, while a few are from 2012 and 2013.

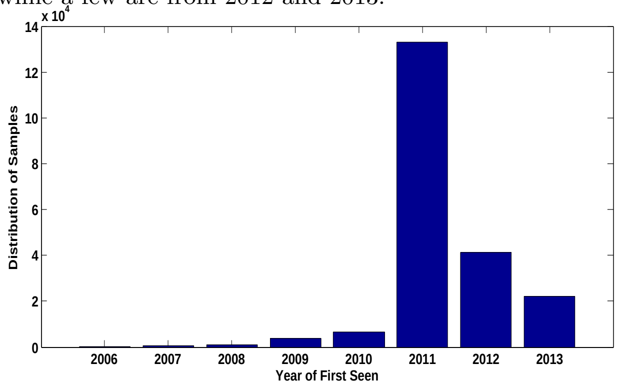


Figure 14: Year of First Seen for the Submitted Samples

File Size: The distribution of the file sizes of various samples are shown in Fig. 15. We see that most of the files have sizes less than 500 kB.

Confidence of Top Match: Of the 212,000 uploaded samples we received, not all the samples have a good match with our corpus database. In Fig. 16, we see the distribution of the confidence levels of the top match. Close to 37% fall under Very High Confidence, 8% under High Confidence, 49.5% under Low confidence and 5.5% under Very Low Confidence. This means that nearly 45% of the uploaded samples (close to 95,400) are possible variants of samples already existing in our database.

AV Label Match vs Confidence Level: Here, we further validate our system by comparing the output of

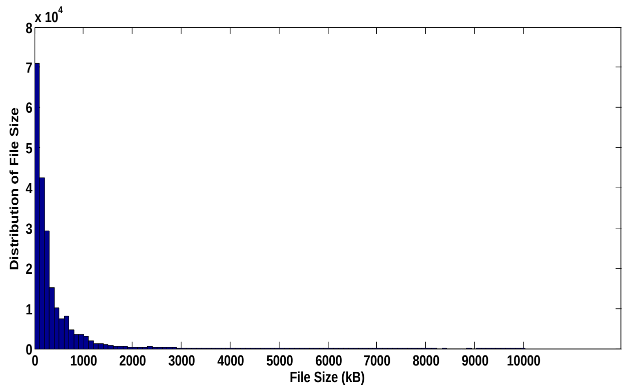


Figure 15: File Sizes of Uploaded Samples (kB)

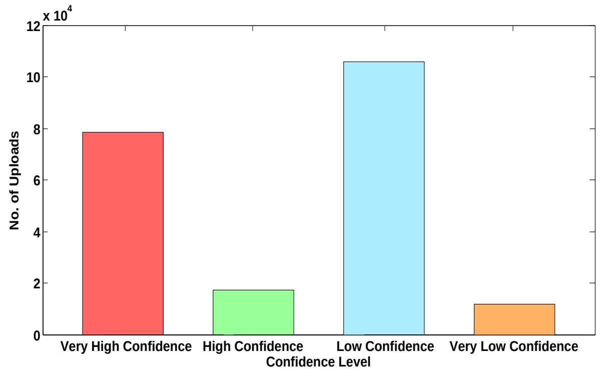


Figure 16: Confidence of the Top Match

our algorithm with the AV labels. For this, we obtained the AV labels for all the uploaded samples and their top match. However, Virustotal has a bandwidth limitation on the total number of samples that can be uploaded. Due to this, we were only able to obtain valid AV labels for a subset of the uploaded samples. We also exclude the uploaded samples that were already present in our database. The labels are then compared as per the methodology in Sec. 3.3.2 and the percentage of correct match is computed. Fig. 17 shows the sorted histogram of the distance between the uploaded samples and their top match. Similar to the results obtained in our earlier design experiment (Fig. 9), we see that the percentage of correct match is high for a low distance. In Fig. 18, we plot this distance versus the percentage of correct match and see that the trend is similar. However, there are a few cases which have a low percentage of correct match for a low distance. This is because we do a one-one comparison of AV labels and malware variants may sometime have different AV labels. For example, the variants *459d5f31810de899f7a4b37837e67763* and *fa8d3ed38f5f28368db4906cb405a503*, we saw earlier in Fig. 10, have AV labels *Trojan.Win32.Refroso.depy* and *Trojan.Win32.Refroso.deqg* as labeled by *Kaspersky* AV vendor. Although, these labels differ only in a character, we do not consider this in our current analysis and these could result in a low percentage of correct match despite having a low distance.

Confidence vs Year of First Seen: For all the uploaded samples, we obtain the year that it was first seen in the wild from Virustotal and compare it with the Nearest Neighbor (NN) distance *d*. In Fig. 19, we plot the year of first seen and the NN distance. We observe that most samples were first seen in the wild between the years 2010 and 2013. Many samples from 2012 and 2013 have a low NN distance and this shows that our system has good matches even with most recent malware. If we consider only the very high confidence and high confidence matches and analyze their year of first seen (Fig. 20), we observe that a large number of samples are from 2011 and a reasonable amount are from 2012 and 2013.

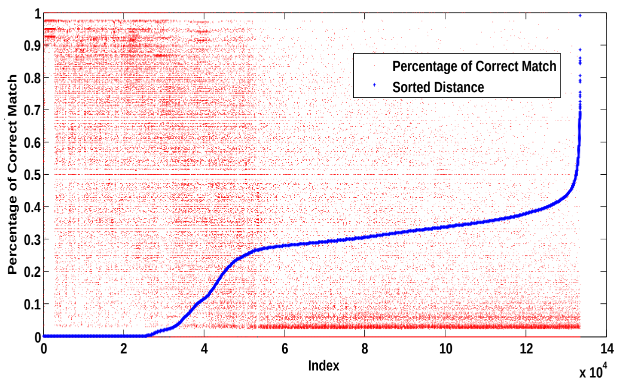


Figure 17: For every uploaded sample, the distances of the top match are sorted (marked in blue) and the corresponding percentage of correct match (marked in red) is overlaid.

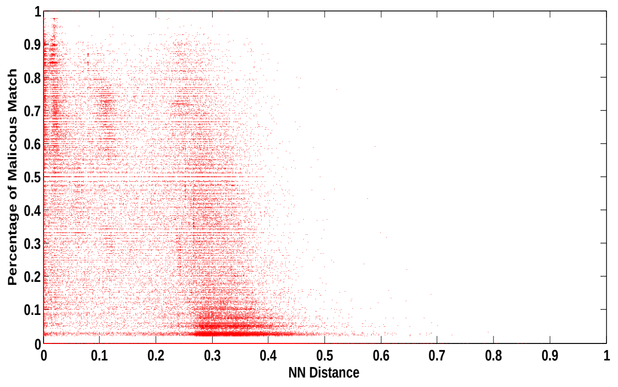


Figure 18: Distance vs Percentage of Correct Match

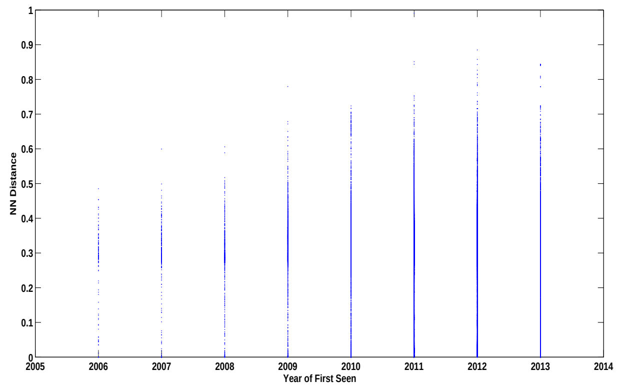


Figure 19: Year of First Seen vs Distance

Packed Samples: We also analyze the performance of SARVAM on packed malware samples. One problem that arises here is that identifying whether an executable is packed or not is not easy. In this analysis, we use the packer identifiers *f-prot* and *peid* that are available from the Virustotal reports. Only 39,260 samples had valid *f-prot* packer sig-

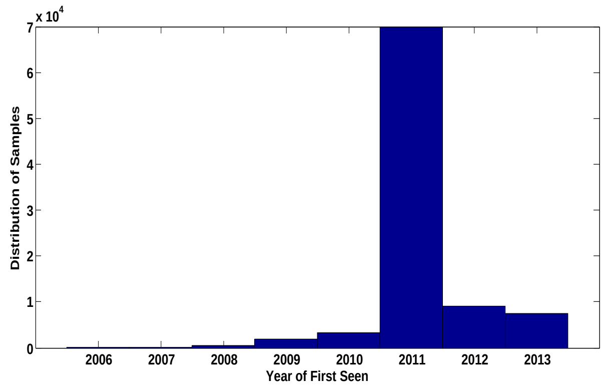


Figure 20: Year of First Seen for Very High Confidence and High Confidence Matches

natures and 49,333 samples had valid *peid* signatures. The actual number of packed samples is usually more but we consider only these samples in our analysis. Of these, 16,055 samples were common between the two and there were 970 unique *f-prot* signatures and 275 unique *peid* signatures in the two sets. This shows the variation in the signatures of these two packer identifiers. For both cases, the most common signature was *UPX*. Others included *Aspack*, *Armadillo*, *Bobsoft Mini Delphi* and *PECompact*. The signature *BobSoft Mini Delphi* need not always correspond to a packed sample and it could just mean that the sample was compiled using *Delphi* compiler. For both sets of samples, we obtain their NN distance and plot the sorted distance in Fig. 21. We observe that nearly half the samples in both sets fall in the Very High Confidence and High Confidence range.

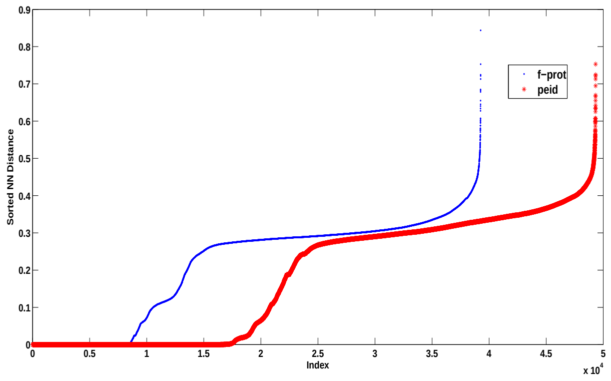


Figure 21: Sorted NN Distance of Packed Samples

Next, we consider only packed samples that fall in the Very high Confidence and High Confidence range (NN distance $d <= 0.25$). This reduced the samples identified by *f-prot* to 14,936 and *peid* to 24,098. Tab. 3 shows the top 5 packer signatures of *f-prot*, with the total number of unique signatures being 364. *UPX* was the most common signature while others included *Allaple*, *PECompact* and *Aspack*. In the case of *peid*, the number of unique signatures was 186. The top 5 are shown in Tab. 4. Again, *UPX* was the most common followed by *Armadillo*, *Bobsoft Mini Delphi* and *PECompact*. This analysis further shows that our approach works on different types of packed samples as well.

5. LIMITATIONS AND FUTURE WORK

One limitation of our approach is that the image fingerprint is computed on the entire executable. Because of this, an attacker can insert invalid codes between sections or interchange the order of the sections which may result in a completely different fingerprint. This could be potentially addressed by analyzing sections of the code rather than the entire code, and computing localized fingerprints. This needs to be further explored. Another limitation is that our system, at present, has not been parallelized and we will look into parallelization techniques in future. Although our query time is less than 3 seconds due to fast nearest neighbor methods such as Balltree, there are faster methods like MinHash, Locality Sensitive Hashing which can reduce the query times much further. Soon, we will be increasing our database to 10 million samples (including mobile malware) and we will consider the above issues while building the next stages of our system. Finally, our approach can only identify malware variants that are similar in structure and cannot identify functionally similar variants that perform the same function but have different structures.

Table 3: Top Packer Signatures of *f-prot*

Packer	No. of samples
UPX	7401
Allaple	920
PecBundle, PECompact	916
Aspack	896
UPX LZMA	724

Table 4: Top Packer Signatures of *peid*

Packer	No. of samples
UPX 2.90 [LZMA]	9285
Armadillo v.1.71	4855
BobSoft Mini Delphi	1537
Armadillo v1.xx - v2.xx	1456
PECompact 2.xx	1402

6. RELATED WORK

Our system is one of the few existing public systems where users can upload malware samples and obtain reports. Other similar systems that let users upload malware include Virustotal [4], Anubis [1] and Malwr [2]. However, while the above systems do static and/or dynamic analysis on a malware sample, ours is the only existing system, to the best of our knowledge, that finds similar malware for an uploaded sample. Other related systems from the context of malware similarity and information retrieval include VILO [15] and NEO [25].

If we consider the literature in malware similarity, most of the works are based on static analysis and a few on dynamic analysis. The static analysis methods can further be classified into techniques based on N-grams [5, 9, 10, 13, 21], N-perms [11, 15], image similarity [12, 17, 18], PE file structure [22,26,27,29] and graph-based methods such as function call graphs or control flow graphs [8, 10, 14].

7. CONCLUSION

In this paper, we presented SARVAM, a system for content-based Search And RetrieVAl of Malware that finds similar malware based on image similarity. SARVAM has been operational since May 2012. During this period, we received close to 212,000 samples of which nearly 45% were possible variants of already existing malware from our database. SARVAM lets users to upload malware samples and obtain the possible variants. Our system has been built on a single desktop computer and the average query time is less than 3 seconds. We are currently working on expanding the database of malware significantly, and including mobile malware.

8. ACKNOWLEDGMENTS

This work is supported by the Office of Naval Research (ONR) under grant N00014-11-10111. We would like to thank Gautam Korlam for his help in the web design of the system.

9. REFERENCES

[1] Anubis. http://anubis.iseclab.org.

[2] Malwr. http://malwr.com.

[3] Offensive Computing. http://offensivecomputing.net.

[4] VirusTotal. http://www.virustotal.com.

[5] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Proc. of the 28th Annual Computer Software and Applications Conference*, volume 2, pages 41–42. IEEE, 2004.

[6] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proc. of the Symp. on Network and Distributed System Security (NDSS)*, 2009.

[7] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid. Evaluation of gist descriptors for web-scale image search. In *Proc. of the ACM International Conference on Image and Video Retrieval*, page 19. ACM, 2009.

[8] X. Hu, T. Chiueh, and K. Shin. Large-scale malware indexing using function-call graphs. In *Proc. of the 16th ACM conference on Computer and communications security*, pages 611–620. ACM, 2009.

[9] G. Jacob, P. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna. A static, packer-agnostic filter to detect similar malware sample. In *Proc. of the 9th Conference on Detection of Intrusions and Malware and Vulnerability Assessment*. Springer, 2012.

[10] J. Jang, D. Brumley, and S. Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proc. of the 18th ACM conference on Computer and communications security*, pages 309–320. ACM, 2011.

[11] M. Karim, A. Walenstein, A. Lakhotia, and L. Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1):13–23, 2005.

[12] D. Kirat, L. Nataraj, G. Vigna, and B. Manjunath. Sigmal: A static signal processing based malware triage. In *Proc. of the 29th Annual Computer Security Applications Conference (ACSAC)*, Dec 2013.

[13] J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2721–2744, 2006.

[14] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *Proc. of the Recent Advances in Intrusion Detection*, pages 207–226. Springer, 2006.

[15] A. Lakhotia, A. Walenstein, C. Miles, and A. Singh. Vilo: a rapid learning nearest-neighbor classifier for malware triage. *Journal of Computer Virology and Hacking Techniques*, 9(3):109–123, 2013.

[16] B. S. Manjunath and W. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI - Special issue on Digital Libraries)*, 18(8):837–42, Aug 1996.

[17] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: visualization and automatic classification. In *Proc. of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, pages 4:1–4:7, New York, NY, USA, 2011. ACM.

[18] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Proc. of the 4th ACM workshop on Security and Artificial Intelligence*, AISec '11, pages 21–30, New York, NY, USA, 2011. ACM.

[19] A. Olivia and A. Torralba. Modeling the shape of a scene: a holistic representation of the spatial envelope. *Intl. Journal of Computer Vision*, 42(3):145–175, 2001.

[20] S. M. Omohundro. Five balltree construction algorithms. Technical report, 1989.

[21] R. Perdisci and A. Lanzi. McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. *Computer Security Applications*, pages 301–310, Dec. 2008.

[22] K. Raman. Selecting features to classify malware. Technical report, 2012.

[23] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. Technical report, University of Mannheim, 2009.

[24] P. Salembier and T. Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[25] I. Santos, X. Ugarte-Pedrero, F. Brezo, P. G. Bringas, and J. M. G. Hidalgo. Noa: An information retrieval based malware detection system. *Computing and Informatics*, 32(1):145–174, 2013.

[26] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo. Data mining methods for detection of new malicious executables. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 38–49. IEEE, 2001.

[27] M. Shafiq, S. Tabish, F. Mirza, and M. Farooq. Pe-miner: Mining structural information to detect malicious executables in realtime. In *Proc. of the Recent Advances in Intrusion Detection*, pages 121–141. Springer, 2009.

[28] A. Torralba, K. Murphy, W. Freeman, and M. Rubin. Context-based vision systems for place and object recognition. In *Proc. of the Intl. Conference on Computer Vision*, 2003.

[29] G. Wicherski. pehash: A novel approach to fast malware clustering. In *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.