Malware Variant Detection Using Similarity Search over Content Fingerprint

Ban Xiaofang[1], Chen Li[1], Hu Weihua[1], Wu Qu[2, 3*]

1. China Information Technology Security Evaluation Center, Beijing 100085
E-mail: banxiaofang@sina.com

2. Tsinghua University, Beijing 100084, China
E-mail: quwu@venustech.com.cn

3. Core Research Institute, Beijing Venustech Cybervision Co. Ltd., Beijing 100193, China

Abstract: Detection of polymorphic malware variants plays an important role to improve information system security. Traditional static/dynamic analysis technologies have shown to be an effective characteristic that represents polymorphic malware instances. While these approaches demonstrate promise, they are themselves subject to a growing array of countermeasures that increase the cost of capturing these malware code features. Further, feature extraction requires a time investment per malware that does not scale well to the daily volume of malwares being reported by those who diligently collect malware. In this paper, we propose a similarity search of malware using novel distance (similarity) metrics of malware content fingerprint based on the locality-sensitive hashing (LSH) schemes. We describe a malware by the binary content of the malware contains; the next step is to compute an feature fingerprint for the malware binary image sample by using the SURF algorithm, and then do fast fingerprint matching with the LSH from malware code corpus to return the top most visually (structurally) similar variants. The LSH algorithm that captures malware similarity is based on image similarity. We implement B2M (Binary mapping to image) algorithm, the SURF algorithm and the LSH algorithm in a complete malware variant detection system. The evaluation shows that our approach is highly effective in terms of response time and malware variant detection.

Key Words: Malware Variant Detection, Content Fingerprint, Locality-sensitive Hashing, Similarity Search

1 INTRODUCTION

Tens of thousands of potentially malware codes are submitted for analysis to network security companies on a daily basis. In 2010, Symantec reported its 2010 corpus at over 286 million [1]. To deal with these vast amounts of malware codes in a real-time manner, autonomous systems for malware code and its variants detection, identification and categorization are required. However, in practice automated detection of malware is hindered by binary obfuscation techniques such as packing or encryption of the executable binary code, and that network attacks frequently design new malware to evade pattern-based detection by anti-virus products.

For each suspected sample a network security company receives, it has to be determined whether the sample is malicious or has been encountered before, possibly in a modified form. The ability to correctly identify commonalities and differences among malware codes would help improve anti-virus products to proactively detect known malware and variant. To facilitate the recognition of similar malwares or commonalities among multiple malwares which have been subject to change, a high-level structure, i.e. an abstraction, of the malwares is required. One such abstraction is also called as content fingerprint. A content fingerprint is a gray scale image representation of a binary executable malware. Packed malware binary code into image is introduced for the first time in the literature [2] for malware classification. The approach borrows techniques from the image processing domain to cast the structure of packed binary samples into two-dimensional gray scale images, and then uses features of this image for classification. Nataraj et al. considered that significant visual similarities in image texture for malware belonging to the same family. This perhaps could be explained by the common practice of reusing the code to create new malware variants.

In this paper, malware code content fingerprint mapping, image features extraction and matching are combined with three techniques, Binary mapping to image algorithm (B2M), speeded up robust features algorithm (SURF), and locality-sensitive hashing algorithm (LSH), intending to migrate similarity computing to hash collision with locality sensitive hashing algorithm and improve the efficiency of malware variant detection based on CBIR (Content fingerprint based image retrieval) significantly. The rest of the paper is organized as follows. Section 2 discusses the related work in malware detection. Related algorithms and techniques of the malware variant detection system (MVD) are discussed in Section 3. Section 4 presents our experimental results and analysis. Section 5 presents a brief description of future work. Finally, the conclusions are drawn in Section 6.

2 RELATED WORK

The paper is related to malware detection, binary data visualization, feature extraction and locality sensitive hashing. The above four parts are briefly discussed in this section.

2.1 Malware Detection

Malware includes viruses, worms, Trojan horses, time and logic bombs, botnets, and spyware. A number of techniques have been devised by researchers to detect these attacks.

978-1-4799-3708-0/14/$31.00 ©2014 IEEE

Traditional signature-based malware detectors identify malware by scanning suspicious binaries for distinguishing byte sequences or features. Features unique to malware are maintained in a signature database, which must be continually updated as new malware is discovered and analyzed. Traditionally, signature database have been manually derived, updated, and disseminated by human experts as new malware appears and is analyzed. However, the escalating rate of new malware appearances and the advent of self-mutating, polymorphic malware over the past decade have made manual signature updating unrealistic. This has led to the development of automated machine learning techniques for malware detection [3-8] that are capable of automatically detection unknown malware.

In terms of feature, there are two main types of features that are commonly used in automated malware analysis [9]: static features based on the malware binary code and dynamic features based on the runtime behavior of the malware. One challenge faced by static-feature analysis techniques is wide-spread introduction of binary obfuscation techniques that are universally adopted by mainstream malware developers. Binary obfuscation, particularly in the form of binary packing, transforms the binary from its native representation of the original source code, into a self-compressed and uniquely structured binary file, such that naive BinDiff-style analyses across packed instances simply do not work. Packing further wraps the original code into a protective shell of logic that is designed to resist reverse engineering, making our best efforts at unpacking and static analysis an expensive, and sometimes unreliable. Dynamic features based on behavioral analysis techniques offer an alternate approach to malware detection. For example, sandboxes may be used to build a detection of binaries based on runtime system call traces [10], or one may detect through host and network forensic analyses of the malware as it operates [11]. Such techniques have been studied as a possible solution, but they may be challenged by a variety of countermeasures designed to produce unreliable results [12]. Dynamic analysis has also been noted to be less than robust when exposed to large dataset [13].

2.2 Binary Data Visualization

Several editor tools such as text editors and binary editors can both visualize and manipulate binary data. In recent years, there have been several GUI-based tools which facilitate comparison of files. However, there has been relatively little research in visualizing malware. In the literature [14], Yoo used self-organizing maps to visualize and detect malware inside an executable binary code. In the literature [15], Quist and Liebrock proposed a visualization framework for reverse engineering. They identify functional areas and de-obfuscate through a node-link visualization where nodes represent the address and links represent state transitions between addresses. In the literature [16], Goodall et al. designed a visual analysis environment that can aid software developers to understand the code better. They also show how vulnerabilities within software can be visualized in their environment. In the literature [17], Conti et al. show that they can automatically classify the different binary fragments using statistical features. In the literature [18], Nataraj et al. proposed a simple yet effective method for visualizing and classifying malware using image processing techniques. Malware binaries are visualized as gray-scale images, with the observation that for many malware families, the images belonging to the same family appear very similar in layout and texture. These works present a similar approach by representing malware as gray scale images.

2.3 Feature Extraction

Searching large-scale image datasets containing millions of images using brute-force matching is not a practical task. Most state-of-the-art large-scale image retrieval approaches use SIFT feature descriptors to represent images. The SIFT was able to be stable with light, noise, and small perspective change in a sense. But SIFT was so complex that it cost too much time and this led to several other improvements, such as principle component analysis SIFT which speed up feature matching by reducing the dimension of image features and fast approximated SIFT which speed up by using an integral image and an integral orientation histogram. Another considerable algorithm on CBIR is speeded up robust features (SURF) [19] and it is stable and fast enough to gain excellent results on region of computer vision such as object recognition and 3D reconstruction. SURF is improved from SIFT both are based on robust points which are not sensitive to transformation, brightness, and noise but is less complex and more efficient than SIFT due to the smaller number and lower dimension of descriptors. An open-source library OpenSURF (C++) written by Dirk-JanK for SURF descriptor extraction is used in this paper.

2.4 Locality Sensitive Hashing

It is the actual situation that SURF descriptors are in 64 dimensions and matching features means searching in high dimension. Among the mainstream searching methods, locality sensitive hashing (LSH) is the fastest way for indexing and could be faster than other methods for several orders of magnitude in huge scale searching. In addition, LSH is based on probability and is more suitable for non-precise searching. LSH was first introduced by Indyk et al. for approximate nearest neighbor search (ANN) [20]. The main idea is to hash vectors using several hash functions and make sure that for each hashing, the vectors with smaller distances between each other are more likely to collide in probability than that with longer distances.

A LSH family of functions can be defined as follows [20]. A family $H = \{h : s \to U\}$ is said to be locality sensitive if, for any q, function $p(t) = \Pr_H[h(q) = h(v) : q - v = t]$ decreases strictly as t increases. That is the probability of the collision of q and v decreases as the distance between them increases. Stable distribution is one of the most

important methods for LSH function implementation. And Gaussian distribution, one kind of famous stable distribution, is used for LSH function design in this paper. Given k, L, W, suppose that A is an $k \times d$ Gaussian matrix, A_i represents the i row of A, $b \in R^k$ is a random vector, and $b_i \in [W]$, $x \in R^d$; then the hash code of x can be represented as:

$$g(x) = (h_1(x), \ldots, h_k(x)) \quad (1)$$

$$h_i(x) = \frac{A_i x + b_i}{W} \quad (2)$$

$g(x)$ is the concatenation of k hash codes, and it is regularly designed as normal hash function to obtain the final scalar index, such as the following one recommended by Andoni and Indyk in one of his LSH library [10]:

$$g(x) = f(a_i, \ldots, a_k) = ((\sum_{i=1}^{k} r_i' a_i) \bmod prime) \bmod tableSize \quad (3)$$

In formula (3) which r_i' is a random integer, prime equals $2^{32} - 5$, $tableSize$ represents the size of hash table, usually equals to |P|, the size of searching space. b_i in formula (2) is a random factor, and because it can be noticed that A itself is random already, so just set $b_i = 0$. Denominator w in formula (2) represents a segment mapping such that the similar values in numerator can be hashed into the same code for the purpose of neighbor searching and its value represents the segment size. In this paper, we chose w=0.125.

There are two more important parameters that should be determined: k for the number of hash codes should be calculated by each hash function and L for the number of hash functions. From the fact that two similar vectors will collide with the probability greater than or equal to (1-δ) when applying LSH, we get some conditions that k and L should satisfy. Suppose that the distance of a query q and its neighbor v is less than a constant R, and let $P_R = p(R)$ then:

$$\Pr_{g \in \zeta}[g(q) = g(v)] \geq p_R^k \quad (4)$$

And for all L hash tables, the probability that q and v does not collide is no more than $(1 - p_R^k)^L$ that is:

$$1 - (1 - p_R^k)^L \geq 1 - \delta \quad (5)$$

We get better performance if there are less hash tables. So let L be the minimum possible integer and there is:

$$L = floor * \frac{\log^{\delta}}{\log(1 - p_R^k)} \quad (6)$$

Now L is a function of k.

According to Andoni and Indyk[21], the best value of k or L should be tested by sampling. Experimental results show that prefer 5, and let the value of be 7. In this paper, the E^{2}LSH[21] package is used to calculate malware content fingerprint's high-dimensional feature vector hash value, and search the feature vector by using the hash value.

3 DESIGN OF THE MALWARE VARIANT DETECTION SYSTEM

3.1 Overall Design

The overall design of the MVD system is as shown in Figure 1. The workflows are as follow:

(1) Malware code mapping: a given malware binary file (exe) is read as a vector of 8 bit unsigned integers, and then be visualized as a gray scale image .

(2) Feature extraction: feature vector in 64 dimensions are obtained by using the SURF algorithm.

(3) Create index: indexing process involves two stages, respectively the LSH mapping phase and the secondary hash phase.

(4) Feature matching: for each hash codes from last step, search corresponding hash table for match features then collect, remove duplicate and sort. According to database address in the inverted list, the MVD system can get the candidate malware content fingerprint's original feature vectors, and then calculate the accurate Euclidean distance between targeted malware content fingerprint and candidate malware content fingerprint set. According to the query conditions, such as distance threshold, filter and return the query results.

(5) Output results.

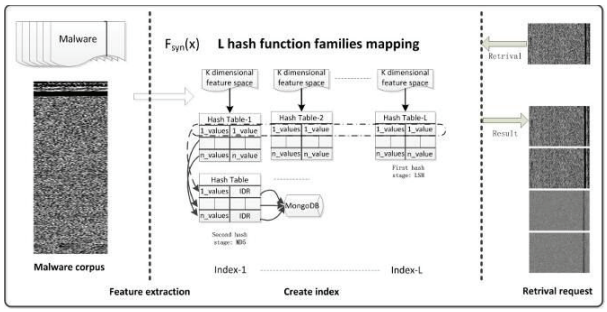

Fig1. The overall design of the malware variant detection system

3.2 Malware Binary File Mapping to Image

A given malware binary file (exe) is read as a vector of 8 bit unsigned integers and then organized into a 2 dimensions array. This can be visualized as a gray scale image in the range [0,255] (0: black, 255: white) [2]. The width of the image is fixed and the height is allowed to vary depending on the file size. The process is also called B2M algorithm in this paper. Figure 2 shows the algorithm process, an example image of a tiger virus, which spreads by using IE aurora zero day vulnerabilities. It is interesting to note that in many cases, different sections (binary fragments) of the malware exhibit distinctive image textures. A detailed taxonomy of various primitive binary fragments and their visualization as gray scale images can be found in [9].

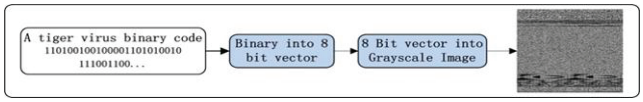

Fig. 2 Image of a tiger virus binary file

3.3 Feature Extraction Based on the SURF

The SURF algorithm is divided into two stages: the robust points' detection and feature description. In the first stage, integral images and fast Hessian matrix is used for detection of image features. In the second stage, a reproducible orientation of each robust point is fixed first, then constructing a square region aligned with the selected orientation and extracting the SURF descriptor with 64 dimensions from this region. Steps are as follow[22]:

(1) Scale spaces analysis: image pyramids are built by repeatedly Gaussian blur and subsampling.

(2) Interest points locating: the maximum of the determinant of the Hessian matrix is calculated first. Then, non-maximum suppression in a 3 × 3 × 3 neighborhood to the image is applied, scale and image space interpolation are taken.

(3) Orientation assignment: haar-wavelet responses in x and y direction in a circular neighborhood around the interest point is calculated. A dominant orientation is estimated and it is now invariant to rotation.

(4) Descriptor extraction: a square region centered around the interest point is constructed and oriented along the orientation selected. Then, the region is split into 4 × 4 square subregions and the sum of wavelet response in horizontal and vertical direction of each subregion is calculated. After normalization, a descriptor in 64 dimensions is obtained.

3.4 Indexing Process Based on the LSH

Indexing process involves two stages, respectively the LSH mapping phase and the secondary hash phase. In the LSH mapping phase, we design the L LSH function families, each LSH family consists of k independent LSH function operations, LSH function operation will map a malware content fingerprint's high-dimensional feature vector to an integer value. After LSH mapping phase (L * k LSH functions), we can get L k-dimensional vectors. According to Andoni and Indyk [21], the best value of k or L should be tested by sampling. Experimental results show that k prefer 5, and let the value of L be 7. In the second hash stage, since this k-dimensional space is sparse, we can use md5 algorithm to efficiently map k-dimensional vector into an integer value. This stage not only reduces the complexity of store and search, but also ensures that the LSH mapping phase of different k-dimensional mapping value still be mapped to different indexes. Each index values constitute a bucket, the image physical address in the MongoDB link to the bucket. At last, the secondary hash stage forms an inverted list. A naive search to find k-dimensional reference points in the same bucket as the query point could easily take O(log N) operations, but we reduce this to O(1) by using md5 hash function.

3.5 Data Structure

There are four kinds of data in the system: the malware binary set, the malware content fingerprint set, the features, and the hash tables. The data are stored in the MongoDB. Details are as follow:

(1) Malware binary set: malware binary and related information are stored in the MongoDB as the JSON structure;

(2) Malware content fingerprint set: malware content fingerprints are stored as image PNG files in the MongoDB as the same as the above JSON, named by using MD5 fingerprint. The algorithm takes as input a filename of arbitrary length and produces as output a 128-bit fingerprint, $ImageId_{key}$;

(3) SURF features table: the vector is in this form: $(ImageId_{key}, Float, Float, \ldots)$. In this feature vector, $ImageId_{key}$ represents the identifier of an image in the MongoDB. Followings are 64 floats, representing a feature vector in 64 dimensions.

(4) Hash table: the hash table is used for approximate nearest query. That is, select a hash code of a specific feature vector as key and query the corresponding value. There are L (L=7) hash tables according to LSH in this paper, so the storage form is similar to SURF features table for the purpose of reducing database connections. That is,

$$LSH\ stage : (ImageId_{key}, ImageFV\_1, \ldots, ImageFV\_i)$$

$$Secondary\ hash\ stage : (ImageId_{key}, (ImageMD_1, \ldots, ImageMD_i)$$

ImageId_key represents the identifier of a malware content fingerprint while i represents the i-th hash function family and $i = 1, \ldots, L$. And $ImageFV\_1$ is a k-dimensional vector by using LSH mapping image SURF features. $ImageMD_i$ is a MD5 hash value by using MD5 algorithm.

4 EXPERIMENTAL RESULTS and ANALYSIS

4.1 Experimental Environment

In this section, the experimental results of the MVD system running on an IBM server are shown and discussed. To examine the availability of the MVD system, we have collected a test dataset consisting of 25 malware families, totaling 8,410 malwares, downloaded from the website: https://zeustracker.abuse.ch/downloads/zeusbinaries.zip.

4.2 Performance Metrics

We evaluate the retrieval performance of the MVD system based on four standard performance metrics: precision, recall, F-measure and query time. The experiment mainly inspects the following four evaluation indexes:

(1) Query Time: refers to the time from the reception of the query to return a result set;

(2) Precision in this paper is the fraction of retrieved malware content fingerprint that are relevant to the targeted malware content fingerprint.

$$precision = \frac{|\{relevant\ malware\ images\} \cap \{retrieved\ malware\ images\}|}{|\{retrieved\ malware\ images\}|}$$

(3) Recall in this paper is the fraction of the malware images that are relevant to the targeted malware image are successfully retrieved.

$$recall = \frac{|\{relevant\ malware\ images\} \cap \{retrieved\ malware\ images\}|}{|\{relevant\ malware\ images\}|}$$

(4) F-measure in this paper is the weighted harmonic mean of precision and recall.

$$F - measure = \frac{2 * precision * recall}{precision + recall}$$

4.3 Experimental Results

To evaluate MVD system, we first try it with the malware "Worm.Whboy", also known as the Panda virus. First, we compute the feature vector, and then query on pre-built hash tables to retrieve top-k ranking similar matches from our database of malware feature vectors with respect to the query. A top-15 query result is as shown figure 3. The experiment result shows that it only takes approximately 35 milliseconds to query the LSH index to obtain the top 15 matches. What we get are the approximate nearest neighbor distances of the query feature vector with other feature vectors in the malware corpus. From the distances, we observe that most of them with the strong similarity. This means that these are variants of the query with very small differences.

To choose a query set for further performance evaluation, we randomly select 8 malwares from each of the 25 malware categories, and form the query set of 200 malware examples. To evaluate how our MVD system performs with respect to different dataset scales, we prepare 10 image datasets of different sizes ranging from 1000 malware content fingerprints to 10000 (8,410 malwares and 1,590 benign binary files, confusion at 15%) malware content fingerprints. We employ the techniques described in section 3.3 to extract the malware content fingerprint's feature vector of the query and malware dataset, in which each malware content fingerprint is represented by a 64-dimensional feature vector. For each of the 10 datasets, we perform two sets of experiments to compare our LSH detection solution with exhaustive linear search method (force method) based on Euclidean distance similarity measure. One set of the experiments employs our proposed LSH detection solution (E^2LSH automatically choose parameters L and k. Experimental results show that k prefer 5, and let the value of L be 7.), while the other set uses a force method. In each experiment, we query the database with a set of query malware images. For each query malware content fingerprint, we retrieve the top 15 ranking malware content fingerprints. The returned ranking malware content fingerprints are checked against the ground truth to figure out the relevant malware content fingerprints. Based on the number of relevant malware content fingerprints, we measure the precision, recall rates and F-measure. The recall and precision rates for all queries are averaged for final performance comparison.

Figure 4(a) shows the experimental results of average precision by using LSH and force respectively in our MVD system. In this figure, top 15 ranking malware images are returned for evaluation. From the figure, we can observe that the results of LSH method is very close to the force method results. Their maximal difference is no more than 3% at any database size. Figure 4 (b) shows the average recall of our MVD system using LSH and force respectively, in which top 15 ranking malware images are returned for evaluation. Figure 5 (a) shows the F-measure of our MVD system using LSH and force respectively, in which top 15 ranking malware images are returned for evaluation. From Figure 4(a), 4(b) and 5(a), we can see that the average recall, average precision and F-measure of our MVD system decrease with the dataset size, yet the decreasing rate diminishes when the dataset size increases. Figure 5(b) shows the average query time performance of our MVD system with LSH versus that with force method. From the results in Figure 5(b), we can see that the query time for force method is linear to the dataset size, while the one for LSH is sub-linear. We can see that the LSH approach is much faster than the force solution with an average speedup greater than 10 times. And the gap of time performance between them grows even faster when the dataset size increases.

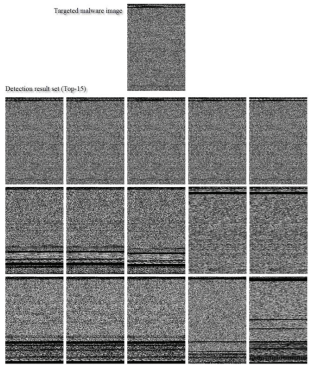

Fig. 3 The malware "Worm.Whboy (Panda virus)" variant detection result

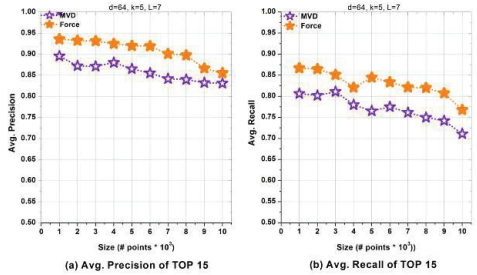

Fig. 4: Graphs showing the average recall and the average precision of the MVD system.

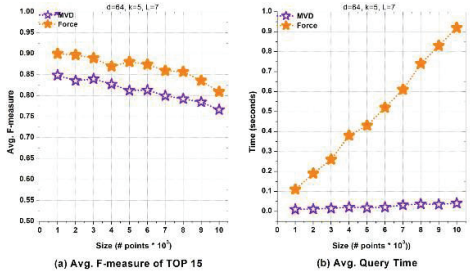

Fig. 5: Graphs showing the average F-measure and the query time of the MVD system.

5 LIMITATIONS and FUTURE WORK

Although a malware content fingerprint retrieval based on the LSH is a novel approach to detect malware variant, an adversary who knows the technique can take countermeasures to beat the system since our technique is based on global image features. Some examples of countermeasures could be relocating sections in a binary or adding vast amount of redundant data. To tackle against such attacks, we will explore more localized feature extraction schemes that take into account the distinct characteristics of malware executable and their primitive binary segments. One possible future extension is to segment out the image regions, and characterize the local texture and spatial distribution of these texture patterns. Another area of future work is parallel solutions, which can further improve the efficiency of our current solution.

6 CONCLUSIONS

In this work, we proposed a novel approach to detect malware variant based on the LSH by processing malware as content fingerprint. A commonly used image feature descriptor SURF is used to characterize the malware globally. The preliminary results on malware variant detection are very encouraging. In this method, we not only search a target malware, but also we search variant (the appropriate nearest neighbor) malwares to a target malware. The proposed approach showed significant improvement, about 100% over original hashing methods. In our future work, we would like to evaluate our method on many other image fingerprint descriptors. Also, we would like to use parallel approaches and larger datasets for exploiting scalability issues.

REFERENCES

[1] D. Takahashi, Symantec identified 286m malware threats in 2010,http://venturebeat.com/2011/04/04/symantecidentified-286m-malware-threats-in-2010/, 2010.

[2] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath. Malware images: Visualization and autmoatic classification, In Proceedings of VizSec, 4, 2011.

[3] KOLTER J, and MALOOF M. A, Learning to detect malicious executables in the wild. In Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (KDD), 470–478, 2004

[4] KOLTER J. Z, and MALOOF, M. A, Using additive expert ensembles to cope with concept drift, In Proceedings of the 22nd International Conference on Machine Learning (ICML), 449–456, 2005.

[5] SCHULTZ, M. G., ESKIN, E., ZADOK, E., and STOLFO, S. J, Data mining methods for detection of new malicious executables. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), 38–49, 2001.

[6] MASUD M. M., GAO J., KHAN L.,HAN J., and THURAISINGHAM B, Mining concept-drifting data stream to detect peer to peer botnet traffic, Richardson, Texas,www.utdallas.edu/mmm058000/reports/UTDCS-05-08, 2008.

[7] MASUD, M. M., GAO, J., KHAN, L., HAN, J., and THURAISINGHAM, B. M, A multi-partition multi-chunk ensemble technique to classify concept-drifting data streams, In Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD), 363–375, 2009.

[8] HAMLEN, K. W.,MOHAN, V.,MASUD, M. M., KHAN, L., and THURAISINGHAM., B. M, Exploiting an antivirus interface, Comput. Stand. Interfaces Vol.31, No.6, 1182–1189, 2009.

[9] Akshmanan Nataraj, Vinod Yegneswaran, Phil Porras, Jian Zhang, A Comparative Assessment of Malware Classification using Binary Texture Analysis and Dynamic Analysis, Workshop on Artificial Intelligence and Security (AISec), Chicago, 21-30, 2011.

[10] C. Willems, T. Holz, and F. Freiling, Toward automated dynamic malware analysis using cwsandbox, IEEE Security and Privacy, Vol. 5, No. 2, 2007.

[11] J. Zhang, P. Porras, and V. Yegneswaran, Host-rx: Automated malware diagnosis based on probabilistic behavior models, Technical report, SRI International, 21-30 , 2009.

[12] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, Impeding malware analysis using conditional code obfuscation, In Proceedings of NDSS, 2008.

[13] P. Li, L. Liu, D. Gao, and M. K. Reiter, On challenges in evaluating malware clustering, In Proceedings of RAID, 238-255, 2010.

[14] Yoo. I, Visualizing Windows Executable Viruses Using Self-Organizing Maps. International Workshop on Visualization for Cyber Security (VizSec), 82-89, 2004.

[15] Quist, D.A. and Liebrock L.M, Visualizing compiled executables for malware analysis. International Workshop on Visualization for Cyber Security (VizSec), 27-32. 2009.

[16] Goodall, J,H. Randwan H. and Halseth. L, Visual analysis of code Security, Proceedings of the Seventh International Symposium on Visualization for Cyber Security, 46-51, 2010.

[17] Conti, G. Bratus, S. Sangster, B. Ragsdale, S. Supan, M. Lichtenberg, A. Perez, R. and Shubina. A, Automated Mapping of Large Binary Objects Using Primitive Fragment Type Classification, Digital Forensics Research Conference (DFRWS), 7, 2010.

[18] Lakshmanan Nataraj, S. Karthikeyan, Gregoire Jacob, B.S. Manjunath, Malware Images: Visualization and Automatic Classification, International Symposium on Visualization for Cyber Security (VizSec), 4, 2011.

[19] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, Computer Vision and Image Understanding, Speeded-Up Robust Features (SURF), Vol.110, No. 3, 346–359, 2008.

[20] P. Indyk and R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, Tex, USA, 604–613, 1998.

[21] A. Andoni and P. Indyk, E^2LSH 0. 1 User Manual, http://www.mit.edu/~andoni/LSH/, 2005.

[22] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding, Vol.110, No.3, 346–359, 2008.