Journal of Cybersecurity Education, Research and Practice

Volume 2020 | Number 2 Article 2

2020

GDOM: Granulometry for the Detection of Obfuscated Malware

John A. Aruta
East Stroudsburg University of PA, jaa8184@gmail.com

N. Paul Schembari
East Stroudsburg University of PA, schembari@esu.edu

GDOM: Granulometry for the Detection of Obfuscated Malware

Abstract

We describe the results of a master's thesis in malware detection and discuss the connection to the learning goals of the project. As part of the thesis, we studied obfuscation of malware, conversion of files into images, image processing, and machine learning, a process of benefit to both the student and faculty.

Malware detection becomes significantly more difficult when the malicious specimen is obfuscated or transformed in an attempt to avoid detection. However, computer files have been shown to exhibit evidence of structure when converted into images, so with image processing filters such as granulometry, it is possible to generate a set of features which will help characterize malicious and non-malicious files. If the structures of file-derived images are resistant to obfuscation, these images may be of valuable use in providing malware signatures. We explore image generated file features and their effectiveness to identify malware when used with various machine learning classifiers.

Keywords

malware detection, obfuscated malware, granulometry, logistic regression, support vector machines, k-means clustering

GDOM: GRANULOMETRY FOR THE DETECTION OF OBFUSCATED MALWARE

1. INTRODUCTION

At East Stroudsburg University, students may complete bachelor's or master's degrees in cybersecurity related fields, and most that complete master's degrees are required to complete a master's thesis. Often, students will find a thesis research topic that interest them on their own, perhaps from reading a journal or proceedings article in one of their courses such as Introduction to Research. The faculty working with these students on their chosen topic then have the benefit of expanding their research knowledge into multiple new subjects. For this reason, we have worked on master's theses in many cybersecurity fields from side-channel attacks on cryptosystems to honeypots to malware detection.

The student learning goals for our thesis requirement are that students gain experience in reading modern research, learning independently, performing some type of experimentation, and writing their results coherently. In this paper we describe the research from a master's thesis on malware detection and we also describe the connection to the learning goals.

2. BACKGROUND AND LITERATURE REVIEW

Since this paper stems from a master's thesis, a literature review was performed, but it was not as extensive as in a doctoral thesis. Overall, twenty-two papers were studied, we worked through a textbook related to our chosen methods, and a few software packages were evaluated for implementation. In general for our master's theses, students are required to find enough sources to make sure that all topics are covered so that the interested reader has the opportunity to gain an understanding of any required background information. To paraphrase one of our now retired faculty: "The literature review should contain the list of sources you wish you had when you started your research." In what follows, for the sake of brevity, we only describe the required background information for our malware detection algorithm.

As is well known, modern computer systems are faced with a staggering number of attack vectors. For example, the website AV-Test.org (AV-Test, 2020) indicated that over one billion malware files were registered by this institute, each a potential attack. When we use the term malware, we include various types of malicious code, such as bots, ransomware, rootkits, scareware, spyware, Trojan horses, viruses, and worms. More specifically, NIST (the US National Institute of Standards and Technology) defines malware as "a program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or otherwise annoying or disrupting the victim." (Souppaya and Scarfone, 2013) A long-established goal of cybersecurity is the detection of malware, which is also the goal of our work.

Because cybersecurity researchers, professionals, and tools attempt to detect malware, such malicious code is often obfuscated. An *obfuscation* is a transformation intended to allow a piece of malicious code into a system without detection. Common obfuscation methods include encryption, packing, or bytecode rearrangement (Sezer, McLaughlin, and O'Kane, 2011). These methods don't change the performance or core functions of the malicious software. As introduced by (Szor, 2005), a system can be actively monitored for any form of deobfuscation code that may execute. However, many files which are not malware are obfuscated to prevent disassembly. Hence, identifying malware based on an actively running unpacker or deobfuscator becomes unrealistic because it can lead to the possibility of extremely high false-positive rates.

The detection of malware is often categorized with regard to two functional methods: static analysis or dynamic analysis. *Static analysis* is performed on a binary while it is not in execution. Static analysis is convenient for malware analysts because it requires minimal resources and also offers quick insight into the capabilities of a possibly malicious file. Static analysis techniques include static string extraction, op code analysis, and control flow analysis, which can be performed with a standard disassembler such as IDA Pro (IDA, 2020). *Dynamic analysis* is performed on a binary during runtime. Control flow can be analyzed during runtime and packed binaries can be manually unpacked. It is important to note that dynamic analysis requires more resources than static analysis, but it can be valuable to malware analysts because it allows them to scrutinize any piece of malware, regardless its degree of obfuscation. Our research involves static analysis in an attempt generate functional signatures for malicious executables. We call our method *GDOM: Granulometry for the Detection of Obfuscated Malware*.

In order to execute GDOM, we use *image processing*. We define an *image* as a function on two variables $f(x, y)$ where (x, y) corresponds to a location (a pixel in a window or printout), and the output of the function represents the "color" at that location. The color may be pure black and white (using 0 and 1), it may be grayscale (using integers 0 – 255), or it may be a three- or four-vector representing RGB or CMYK. *Image processing* concerns transformations performed on digital images. Examples include restoration (restoring noisy images), filtering (extracting properties), and modification. Image filters can provide ideally unique characteristics for a particular image. In our algorithm, we will convert files into images, and then process them with a *granulometry filter*, "a method for characterizing granular images by means of how they are sieved through sieves of various size and shape." (Dougherty and Lotufo, 2003, p. 193)

After our potential malware has been converted into an image file and processed with a granulometry filter, we apply *machine learning* to classify the file as malware or not. Machine learning makes "computers modify or adapt their actions… so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones." (Marsland, 2009) *Supervised learning* is a subset of machine learning that utilizes an algorithm and labeled training set in an attempt to make correct predictions. We can also apply an unsupervised *clustering algorithm* to this type of problem so that the files are separated into two clusters representing malware and not, but training is not necessary. We see that classifying malicious files with machine learning is a *binary classification problem*, where the classification is based on some type of decision boundary creating a partition of the Euclidean space containing our image-represented files into "Malicious" and "NonMalicious".

Hence, the goals of GDOM are:

- Generate malware signatures that are resistant to the obfuscation of their parent binary
- Find evidence of unique structures in files that can be extracted to serve as signatures
- Integrate file signatures with machine learning classifiers or clustering algorithms in order to provide successful malware detection

To summarize our GDOM approach, obfuscated files which are potentially malware are transformed into images and then granulometrically filtered to provide unique characteristics about their parent files. Each potential malware file produces one vector of characteristics, which is the signature of the file. The vectors are then input into several machine learning algorithms and classified as malicious or non-malicious, which also gives the classifications of the parent files. Hence, we explore the effectiveness of the image processing technique of granulometry in relation to malware identification illustrating that it offers some resistance to malware obfuscation.

3. MATHEMATICS, TECHNIQUES, AND ALGORITHM

In this section, we introduce the specific techniques used in our approach, and we fully describe our algorithm. To begin, we decided to work only with Windows x86 executables, but our techniques can be used on any type of file format. Second, our implementation was written in C#, and we used the ImageJ (US National Institute of Health, 2020) and WEKA (University of Waikato, 2020) packages in order to perform image processing and apply our chosen machine learning algorithms, respectively. These selections were made by the student as part of the process of learning how to experiment and create a reasonable data set for research analysis.

3.1 File Selection

To create a set of files, we first downloaded a collection of potential malware from the virus research site VirusSign (VirusSign, 2020). This collection contained 32-bit and 64-bit Windows executables, PDF, and DLL files, but we only used the 32-bit EXEs. To test that these files were actually considered malicious, we extracted them from a ZIP file on a computer running an updated anti-virus system and all files were recognized as malware. Further, to ensure that some of the malicious files were obfuscated, we used PEiD (Aldeid, 2020) to check for packing. In our collection, a variety of files were identified as having been packed by UPX (Ultimate Packer for eXecutables, 2020), PECompact (Bitsum.com, 2020), and ASProtect (Aspack Software, 2020). We should note that having some files obfuscated and some files not obfuscated was an important experimental consideration since if we only used packed malware, the machine learning classifiers might be tailored to detect malware based on packing. We also needed to include non-malicious files in our test collection, so we used 32-bit EXEs from the Windows XP Professional and Windows 7 Professional operating system files for a total of 804 non-malicious and 804 malicious files.

Again, these selections were made by the student as part of the process of learning how to experiment and create a reasonable data set for research analysis.

3.2 Image Creation

In our implementation, we decided to use grayscale images to represent the files, and hence we require a byte for each pixel. Like any computer file, each EXE under consideration can be viewed as a list of bytes, and hence a vector. To convert a file to a pixel representation, we require two coordinates for the location of each byte. Because of this, as we convert files, we need to determine a length and a width of each of the generated images. Following the taxonomy created by (Nataraj, Yegneswaran, Porras, and Zhang, 2011), based on their own empirical observations, we used the widths shown in Table 1 for our file-generated images. Finally, in the event of a remainder, where not enough bytes were available to fill an entire row, we filled out the pixels with bytes of 0, giving intensity 0. Hence, the overall process reads one byte of the file at a time, converts that byte to a grayscale intensity, and then places this intensity into the pixel based on the Nataraj, et al, width methodology. Figure 1 shows a sample file from Nataraj, et al. Notice, that we are using a previously published method for the image generation, again a good technique for students to learn when they are beginning their research programs.

File Size Range (KB)	**Image Width (Pixels)**
Less than 10	32
10 – 30	64
30 – 60	128
60 – 100	256
100 – 200	384
200 – 500	512
500 – 1000	768
More than 1000	1024

Table 1: Image Width Based on File Size

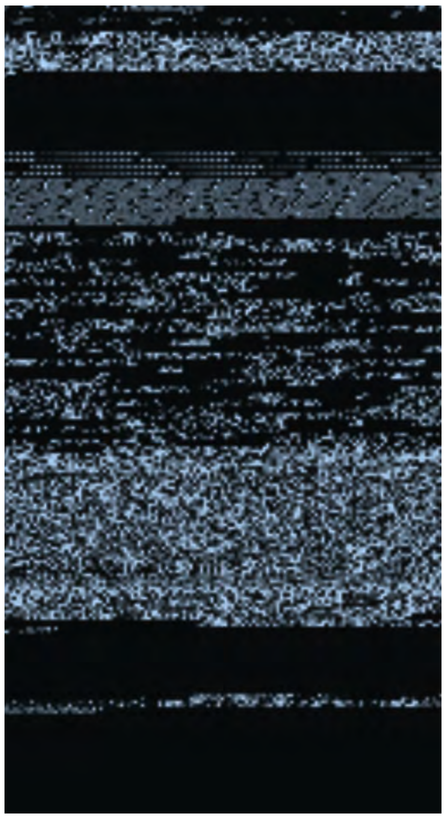

Figure 1: Sample Image File

3.3 Image Processing with Granulometry

Previous researchers have used various image processing techniques to determine file structure and apply this to malware classification. One of the first papers in this area involved the analysis of files as grayscale images by (Conti et al, 2010). In two papers, (Nataraj et al, 2011) and (Nataraj, Karthikeyan, Jacob, and Manjunath, 2011) used a GIST descriptor and a set of Gabor filters, which are commonly used in scene classification, object recognition (Oliva and Torralba, 2011), and texture extraction, to classify files as malware. In order to try a new approach, we decided to use the image processing technique of *granulometry* in GDOM. This is part of the experimentation of the master's thesis, following a yet unused approach.

In order to gain a good understanding of granulometry, we read through Doughetry and Latufo's text, *Hands-on Morphological Image Processing*, as well as worked through many of the exercises. Granulometry uses the mathematical morphology concepts of *translation, erosion, dilation,* and *opening* (Dougherty and Lotufo, 2003, Chapter 1). For a binary (black/white) image A, these operations are defined as follows:

- For a point or vector x, the *translation* of the image A by x is
$$A_x = \{a + x : a \in A\}$$
- For the binary image B, usually called the *structuring element*, the *erosion* of A by B is
$$A \ominus B = \{x : B_x \subset A\}.$$
- The *dilation* of A by the structuring element B is
$$A \oplus B = \{A_b : b \in B\}.$$
- The *opening* of the image A by the structuring element B is
$$A \circ B = (A \ominus B) \oplus B$$

As examples for the above processes, following Dougherty and Lotufo, let us assume the binary image A and the binary structuring element B are represented by the matrices below, where we envision the origin at the center of each matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For erosion, it may be easiest to find the entries of A where the positions of the nonzero entries of the matrix B are found – almost like an intersection. Hence, for erosion in this example, we are looking for the value of 1, where this value is repeated in a position one above and to the right. Hence, we can determine the erosion by proceeding through each point of A, placing the matrix B centered at that point (with respect to B's origin), and if the two values of 1 from B "match" the values in A, we select that point from A to be part of the erosion. When following this process, we can assume that any points outside of the matrix are 0. Dilation, instead, is the dual of erosion – we imagine "unioning" or "adding" the structural element with the original matrix. We proceed through each point of A which is 1, place B centered at that point, and then because of the other number 1 in the matrix B, make sure that 1 also appears in that position of the result. Finally, opening combines the two processes. For this example, the results of erosion and dilation are given below:

$$A \ominus B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad A \oplus B = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For grayscale images, the processes of erosion and dilation are similar if we consider the intensity (0 – 255) as the height of a function at the pixel (x, y), and we think of a solid gray shape under the heights. (We are now thinking of the solid gray shape as the image.) In this case, to perform erosion of a signal, the center of the structuring element (SE) is aligned with the x-coordinates and pushed upward from negative infinity until part of it exits the solid shape of the original signal. The position of the structuring element is marked, and any space above the SE becomes undefined. Note that spaces where the signal is originally undefined play no role in the process of grayscale erosion as there is no intensity to erode. In order to perform dilation of a signal, the center of the structuring element is aligned with the x-coordinates, but now pushed downward from positive infinity until part of it enters the solid shape of the original signal. The position of the structuring element is marked, and any space under the SE becomes part of the new signal. In this case, spaces where the signal is originally undefined can also become part of the transformed signal. Finally, the opening operation is defined in the same manner as the binary case.

As described by (Dougherty and Lotufo, 2003, p. 193), (Matheron and Serra, 2002) defined a granulometric method which uses different size and shape structuring elements to sieve images and thus is effective for texture analysis. This is the method we will use to obtain a "signature" for each potential malware file. To perform this analysis, we used the tool ImageJ (US National Institute of Health, 2020). Specifically, we used the Granulometry Plugin (Prodanov, 2020) which requires images to be in 8-bit grayscale format. To create the signatures for our images, we consider the integer $t > 0$ a variable and define a granulometry of the image A based on the generator (structuring element) B as

$$\Psi_t(A) = A \circ tB = A \circ \Big(\big((B \circ B) \circ B\big) \circ \ldots \circ B\Big),$$

where the opening of B is repeated t times**.** In GDOM, we used a disk as the base structuring element, so tB represents disks of increasing radii t. Then, if we let $v(A)$ represent the area of the image A, we obtain the area removed by the opening by tB, called the *size distribution* of the image:

$$\Omega(t) = v(A) - v(A \circ tB).$$

Notice that for sufficiently large t, $\Omega(t) = v(A)$. Then the *normalized size distribution* is given by

$$\Phi(t) = \frac{\Omega(t)}{\Omega(\infty)} = \frac{\Omega(t)}{v(a)}\ .$$

Finally, we create the image signature, called the *discrete pattern spectrum*, by taking the discrete derivative of

$$d\Phi(t) = \Phi(t+1) - \Phi(t).$$

In GDOM, we allowed the integer *t* to range from 0 up to maximal radius between 10 and 20 with a step size of 1.

3.4 Machine Learning

Once the potential malware files were converted to images, and then summarized via the granulometry signatures, we then classified them as “malicious” or “non-malicious” by using three of the machine learning algorithms built into WEKA (University of Waikato, 2020). In particular, we used the SimpleLogistic function which creates logistic regression models, the LibSVM library for support vector machine classification, and the SimpleKMeans class for clustering data using the $\boldsymbol{k}$-means algorithm.

Overall, our problem involves a set of vectors, the image signatures, and we want to classify them as “malicious” or “non-malicious”. Hence, we require a mapping from $\mathbb{R}^n$ to $\{0,1\}$, where the elements of the domain are typically called feature vectors and the elements of the range are called the labels. To create a Logistic Regression model, we introduce the function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

which has a limit of 0 as $z \to -\infty$, and a limit of 1 as $z \to \infty$. For the input vectors x, and a fixed vector θ, we then create the hypothesis function

$$h_\theta(x) = \sigma(\theta^T x).$$

Using this function, the goal of logistic regression is to find the best θ so that $h_\theta(x)$ is large when x should have a label of 1, and $h_\theta(x)$ is small when x should have a label of 0. This is performed by using the correct cost function (Stanford University, 2017). We find θ by training the classifier on part of our input data.

Then, we test our solution on the remaining data.

We also used the Support Vector Machine (SVM) model as a comparison to Logistic Regression. If we view our feature vectors in $\mathbb{R}^n$ with some of these representing malicious code and the others representing non-malicious code, then we can consider the largest "rectangular" region (we use rectangular in the n-dimentionsal sense), called the *margin*, separating the two types of vectors. The support vectors are the feature vectors which are closest to this margin. The best classifier is a hyperplane which bisects the margin in a perpendicular fashion so that the two types of feature vectors are on opposite sides of the hyperplane. Finding this best classifier is a constrained optimization problem (Marsland, 2009, Chapter 8). As in the logistic regression case, the SVM approach requires training.

The final algorithm that we used for classification, the k-means algorithm [7], does not require training. Again, we view our feature vectors in $\mathbb{R}^n$, but instead we separate the vectors into two clusters by choosing those "closest" to each other. To measure the distance between two vectors, we used the standard Euclidean distance function. As described by (Redmond and Heneghan, 2007), first we choose a random set of two cluster centers, called the *Forgy initialization*. Second, each vector is assigned to a cluster based on its nearest center. Third, each cluster center is updated as the mean of the current set of vectors in the cluster. Last, the error between the vectors and the centers is calculated. The algorithm is repeated until the calculated errors are minimized.

The final technique which we used in GDOM was 10-*fold cross validation*. In this process, the data (our original 1608 files) was divided into 10 randomly selected sets. Nine of the sets were used for training, and the last set was used for actual testing. This process was repeated 9 more times, where each of the 10 sets acted once as the testing set, while the other 9 sets acted as the training sets (Marsland, 2009, pp. 20-21).

As we can see from this description, the thesis has definitely met the goals of allowing the student to experiment and stretch their learning outside the classroom and in an independent fashion.

3.5 Algorithm

With the parts described above, we can now illustrate our algorithm, GDOM: Granulometry for the Detection of Obfuscated Malware.

1. Potential malware files are converted into grayscale images, with widths based on Table 1, by writing each byte as a grayscale intensity. If we do not have enough bytes to fill a row, we use 0 intensity for the missing pixels.
2. The images are probed with disks of increasing radii, creating a granulometry filter. The radii range from 0 up to a maximum radius between 10 and 20, with step size 1.

3. The probing of step 2 creates a file signature, based on the Discrete Pattern Spectrum of the image.
4. The files are then grouped into two clusters to indicate malicious or non-malicious files. This classification is completed with the machine learning methods of Logistic Regression, Support Vector Machines, and k-Means Clustering (with $k = 2$).

4. EXPERIMENTATION AND RESULTS

To illustrate our results by algorithm based on the files as selected in Section 3.1, we present three tables. Table 2 shows the results when processing the signatures under Logistic Regression, where the TPR (True Positive Rate) gives the percentage of malware detected, TNR (True Negative Rate) gives the percentage of non-malware detected, and the Accuracy gives the average of the TPR and TNR. The general trend here is that higher radii give slightly better results. Since the detection of malware is of utmost importance, the TPR is the most significant measure of an algorithm such as GDOM. Hence, we concentrate our analysis on the TPR. Furthermore, it is important to remember that our experimentation included both obfuscated and non-obfuscated files.

Max. Radius	Accuracy	TPR	TNR
10	0.7618	0.8159	0.7077
11	0.7649	0.8172	0.7127
12	0.7631	0.8159	0.7102
13	0.7680	0.8209	0.7152
14	0.7668	0.8197	0.7139
15	0.7624	0.8172	0.7077
16	0.7662	0.8221	0.7102
17	0.7668	0.8221	0.7114
18	0.7693	0.8209	0.7177
19	0.7674	0.8197	0.7152
20	0.7699	0.8221	0.7177

Table 2: Logistic Regression Results

Table 3 shows the results when processing the signatures with the Support Vector Machine method. The general trend here is that higher radii give slightly better results. We have especially good results in the True Positive Rate.

Max. Radius	Accuracy	TPR	TNR
10	0.7419	0.8271	0.6567
11	0.7450	0.8371	0.6530
12	0.7475	0.8420	0.6530
13	0.7481	0.8420	0.6542
14	0.7481	0.8458	0.6505
15	0.7475	0.8470	0.6480
16	0.7475	0.8495	0.6455
17	0.7475	0.8520	0.6430
18	0.7438	0.8520	0.6356
19	0.7552	0.8532	0.6538
20	0.7419	0.8557	0.6281

Table 3: SVM Results

The best results come from the k-Means Clustering algorithm, as shown in Table 4. We see a True Positive Rate of 88% when the maximum radius is 20.

Max. Radius	Accuracy	TPR	TNR
10	0.7239	0.6863	0.7804
11	0.7270	0.8246	0.6294
12	0.7264	0.8296	0.6231
13	0.7295	0.8271	0.6318
14	0.7245	0.8333	0.6157
15	0.7257	0.8358	0.6157
16	0.7276	0.8470	0.6082
17	0.7264	0.8420	0.6107
18	0.7282	0.8495	0.6070
19	0.7264	0.8433	0.6095
20	0.6934	0.8843	0.5025

Table 4: k-Means Results

Scanning Tables 2, 3, and 4, we find the best results (measured by the rates) with regard to Accuracy, TPR, and TNR as shown in Table 5. This indicates that GDOM works best especially with regard to the TPR by using the k-Means Clustering algorithm. It is interesting that the best detection of malware came with a maximum radius of 20, while the best detection of non-malware came with a maximum radius of 10.

Measure	Algorithm	Max. Radius	Rate %
Accuracy	Logistic	20	76.99
TPR	$k-$Mean	20	88.43
TNR	$k-$Mean	10	78.04

Table 5: Best Results by Measure

Since the above work satisfied the learning goals for the thesis, we did not compare these results with methods used by other researchers as part of the thesis. In Table 6 we include some comparisons for the sake of completeness. Blank entries in the table indicate that these results were not reported.

Papers	Accuracy	TPR	TNR
Nataraj et al, (2011) and Nataraj, Karthikeyan, Jacob, and Manjunath (2011)	86% – 98%	---	---
Makandar and Patrot (2015)	90%	90%	---
Kosmidis and Kalloniatis (2017)	85% – 91%	---	---
Kumar, et al (2018)	98%	---	---
Shukla, et al (2019)	94%	---	---

Table 6: Results Reported by Other Researchers

5. CONCLUSION

Our algorithm, experimentation, and analysis indicate that the mathematical morphology method of granulometry is a good tool for the detection of malware, including obfuscated malware. Furthermore, the project clearly meets the learning goals of our thesis requirement. Our experimental results also show evidence that files exhibit identifiable texture patterns when converted to images, and that these patterns can be detected by a granulometry filter. In particular, malware can be identified using the discrete pattern spectrum of the derived image as a signature and the $\boldsymbol{k}$-Means Clustering algorithm to classify the malware. This method does not report results as good as those by other researchers, but enhancements may improve the results. One of the possible enhancements may be to combine the machine learning methods.

Besides enhancements, other open questions remain. First, ImageJ was only capable of performing granulometry filtering with disk shaped elements during the time of feature extraction. It would be valuable to explore the use of different shaped structuring elements. Second, we limited the maximum radii of the disks to values between 10 and 20 in order to provide a reasonable testing scope. Perhaps larger radii would give better results. Third, we increased the radii of the structuring elements by 1 each time. It is possible that a different increase would improve results. Fourth, we used the results of previous researchers to choose the widths of our images. Hence, different image width schemes should be examined. Finally, experimentation should be completed on other file types to determine the efficacy of GDOM.

Despite the fact that some malware was packed, and some wasn't, GDOM obtained very good True Positive Rates for malware detection especially since obfuscated malware is more difficult to identify. The Accuracy and True Negative Rates (detection of non-malware) of GDOM are also promising and show evidence that transforming files into images, subsequently generating signatures for the images using granulometry, and using the signatures as input for machine learning algorithms serves as a valuable approach to distinguish malware and non-malware whether or not files are obfuscated. Finally, it is clear from our work on this thesis that undertaking a project of this kind is of definite benefit to both student and faculty.

REFERENCES

Aldeid.com (2020), *PEiD*. https://www.aldeid.com/wiki/PEiD

Aspack Software (2020), *ASProtect 32 - application protection tools for software developers*. http://www.aspack.com/asprotect32.html

AV-Test (2017): *Malware statistics and trends report*. www.av-test.org/en/statistics/malware/

Bitsum.com (2017), *PECompact - windows (PE) executable compressor*. https://bitsum.com/portfolio/pecompact/

Conti, G., et al (2010), *A visual study of primitive binary fragment types*. http://www.rumint.org/gregconti/publications/taxonomy-bh.pdf

Dougherty, E.R. & Lotufo, R.A. (2003), Hands-on Morphological Image Processing. Washington, USA: SPIE Press.

IDA: About (2020). www.hex-rays.com/products/ida/

Kosmidis, K. and Kalloniatis, C. (2017), *Machine Learning and Images for Malware Detection and Classification*. Proceedings of the 21st Pan-Hellenic Conference on Informatics, pp. 1–6.

Kumar, R., Xiaosong, Z., Khan, R. U., Ahad, I., Kumar, J. (2018), *Malicious Code Detection based on Image Processing Using Deep Learning*. Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 81–85.

MacQueen, J. (1967), *Some methods for classification and analysis of multivariate observations*. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, pp. 281–297. California, USA: University of California Press

Makandar, Aziz and Patrot, Anita (2015), *Malware Image Analysis and Classification using Support Vector Machine*. International Journal of Advanced Trends in Computer Science and Engineering, Vol.4, No.5, pp. 01-03.

Marsland, S. (2009), Machine Learning: An Algorithmic Perspective. London, UK: Chapman and

Hall/CRC.

Matheron, G. & Serra, J. (2002), *The birth of mathematical morphology*. Proceedings of the Sixth International Symposium on Mathematical Morphology, pp. 1–16. Sydney, AU: Chapman and Hall/CRC

Nataraj, L., Karthikeyan, S., & Jacob, G., Manjunath, B.S. (2011), *Malware images: Visualization and automatic classification*. VizSec 2011, pp. 4:1–4:7. New York, US: ACM.

Nataraj, L., Yegneswaran, V., Porras, P., & Zhang, J. (2011), *A comparative assessment of malware classification using binary texture analysis and dynamic analysis*. Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, pp. 21–30. New York, US: ACM.

Oliva, A. & Torralba, A. (2001), *Modeling the shape of the scene: a holistic representation of the spatial envelope*. International Journal of Computer Vision 42(3), pp. 145–175. New York, US: Springer.

Prodanov, D. (2020), *Granulometry plugin for ImageJ*. https://imagej.nih.gov/ij/plugins/granulometry.html

Redmond, S.J. & Heneghan, C. (2007), *A method for initialising the k-means clustering algorithm using kd-trees*. Pattern Recognition Letters 28(8), pp. 965–973. Edinburgh UK: Elsevier.

Sezer, S., McLaughlin, K. & O'Kane, P. (2011), *Obfuscation: The hidden malware*. IEEE Security and Privacy Volume: 9 (Issue: 5, Sept.-Oct. 2011) pp. 41–47. New York, US: IEEE.

Shukla, S., Kolhe, G., Manoj P D, S., and Rafatirad S. (2019), *MicroArchitectural events and image processing-based hybrid approach for robust malware detection: work-in-progress*. Proceedings of the International Conference on Compliers, Architectures and Synthesis for Embedded Systems Companion, pp. 1–2.

Souppaya, M. & Scarfone, K. (2013), *Guide to malware incident prevention and handling for desktops and laptops*. nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-83r1.pdf

Stanford University (2020), Department of Computer Science: *Deep learning tutorial - logistic regression*. http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression/

Szor, P. (2005), The Art of Virus Research and Defense, Antivirus Defense Techniques. New Jersey, US: Addison-Wesley Professional.

Ultimate Packer for eXecutables (2020), *UPX: Ultimate Packer for eXecutables*. https://github.com/upx/upx

University of Waikato (2020), *WEKA3*. http://www.cs.waikato.ac.nz/ml/weka/

US National Institute of Health (2020): *ImageJ*. https://imagej.nih.gov/ij/

VirusSign (2020), *VirusSign*. http://www.virussign.com/