Society for Science and Education
United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
ISSN 2054-7390

Application of Hybrid Machine Learning to Detect and Remove Malware

[1]**Richard R. Yang**, [2]**Victor Kang**, [3]**Sami Albouq and** [3]**Mohamed A. Zohdy**
[1]*College of Engineering and Applied Sciences, University of Wyoming, United States;*
[2]*College of Literature, Science and the Arts, University of Michigan Ann Arbor, United States;*
[3]*School of Engineering and Computer Science, Oakland University, United States;*
ryang3@uwyo.edu; vkang@umich.edu; salbouq@oakland.edu; zohdyma@oakland.edu

ABSTRACT

Anti-malware software traditionally employ methods of signature-based and heuristic-based detection. These detection systems need to be manually updated with new behaviors to detect new, unknown, or adapted malware. Our goal is to create a new malware detection solution that will serve three purposes: to automatically identify and classify unknown files on a spectrum of malware severity; to introduce a hybrid machine learning approach to detect modified malware traces; and to increase the accuracy of detection results.

Our solution is accomplished through the use of data mining and machine learning concepts and algorithms. We perform two types of data mining on samples, extracting n-grams and PE features that are used for our machine learning environment. We also introduce a new hybrid learning approach that utilizes both supervised and unsupervised machine learning in a two-layer protocol. A supervised algorithm is applied to classify if a file is considered malware or benign. The files classified as malware will then be categorized and then assigned on a severity spectrum using the SOFM unsupervised algorithm.

Keywords: machine learning, data mining, self-organizing feature map algorithm, malware detection.

1 Introduction

Due to the advancements and increased use in technology, malware is becoming more prevalent in today's society. Malware are widely categorized, but the main types we investigate are: Adware, Trojans, Keyloggers, and Rootkits. The purpose of Adware is to obtain users internet behavior for commercial purposes such as targeted ads. A Trojan disguises itself as legitimate or useful software but actually infiltrates a user's system. Keyloggers record keystrokes and other user input to a system. Finally, a Rootkit attempts to gain elevated control of a user's system while hiding itself and other system components. Each category of malware seeks to accomplish a different purpose, but their overarching goal is to cause malicious intent and to steal information from users without their consent.

Currently, anti-malware solutions are able to fight back new malware through two approaches: signature-based detection and heuristic-based detection. In signature-based detection, a suspicious file is obtained by the anti-malware firm and analyzed. If the file is deemed to be malware, a unique signature is generated for the file and that signature is added to an online database. The anti-malware solution is able to quarantine files that match malicious signatures on clients systems. Although this method has been

DOI: 10.14738/tmlai.34.1436
Publication Date: 26th August, 2015
URL: http://dx.doi.org/10.14738/tmlai.34.1436

reliable and commonly used, the downside is that anti-malware firms must readily obtain samples and manually update the database each time new or modified malware surface. Compared to signature-based detection, heuristic-based detection generalizes malware by using a single signature for any of the malware's variations or mutations. However, systems that use heuristic-based detection often has a high number of false positives [3]. Though these methods have been seen to be effective, we choose to investigate a machine learning alternative to these current detection methods.

The two primary categories of machine learning are supervised learning and unsupervised learning. In supervised learning, the system is given a set of training data and how that data is classified. The algorithm is able to classify new input by learning from the training set that it was provided. The results from supervised learning are more specific than its counterpart. In unsupervised learning, the system is provided only with data and no classification keys. The idea of unsupervised learning is to classify relatively similar inputs together without any prior classification knowledge about the data set.

In our research, we expand the application of data mining and machine learning concepts for malware detection and introduce a new hybrid machine learning approach that incorporates both supervised and unsupervised learning in a two-step classification process.

2 Related Work

Alternatives to signature-based detection and heuristic-based detection in security software have been proposed and examined in the past [2] [5] [6] [7] [10].

In [7], the authors use data mining and natural language processing on End-User License Agreements (EULA) during software installation. The authors' objective were to detect spyware that may have been installed with the user's consent through the EULA, where the EULA may have been written in a way that is hard to comprehend for users. The study compared 17 learning algorithms with a baseline algorithm on a bag-of-words and meta-data model of representing the EULA.

Another method of detection was introduced by [10], where a Support Vector Machine is used in conjunction with static and dynamic analysis of file samples. Static analysis of samples were performed by dumping the program's import table from the Portable Executable structure, and recording all of the DLLs used and API function calls. For dynamic analysis, the authors ran each sample in a VMWare sandbox environment and monitored changes in registry, system folders/files, and network states. The results of each type of analysis were used as features in the Support Vector Machine, and information gain was calculated to observe the influence of feature selection.

A system known as Early Detection, Alert and Response (eDare) was designed to detect malicious code in network traffic in [6]. The eDare system uses network traffic scanners and machine learning algorithms to "pinpoint unknown malicious code exhibiting suspicious morphological patterns." [6] In this investigation, static code analysis is performed using decision trees, neural networks, and Bayesian networks.

Unsupervised learning was introduced to detecting malware in wireless devices by [2]. In [2], k-Means Clustering, γ-Algorithm, Divisive Hierarchical Clustering, Agglomerative Hierarchical Clustering, and Quartersphere Support Vector Machine unsupervised learning algorithms were applied to features data mined from Android applications.

The approach of using data mining and supervised machine learning was investigated in [5]. The authors extract n-grams from file samples and apply supervised learning algorithms to classify files as purely spyware or benign. N-grams of various sizes (centered on n = 5) and several supervised learning algorithms were compared to create a better performing solution than existing anti-virus software.

3 Experimental Procedure

Our experimental procedure begins with collecting benign and malware samples from several trusted websites. Next, we data mine and extract features from the samples. A feature is an identifying characteristic of the file used in machine learning algorithms. In our research, we use two general types of features: N-grams and Portable Executable structure. After data mining and obtaining features from our samples, we proceed with our hybrid machine learning approach. First, the features are organized in a database and we run it through supervised learning algorithms that will strictly classify if a file is malware or benign. From there, we use all of the files that were identified as malware and use them as input to an unsupervised learning algorithm to classify the malware on a severity spectrum. Finally, we evaluate our approach by computing the true positive rate, false positive rate, overall accuracy, and area under the receiver operating characteristics curve for each trial.

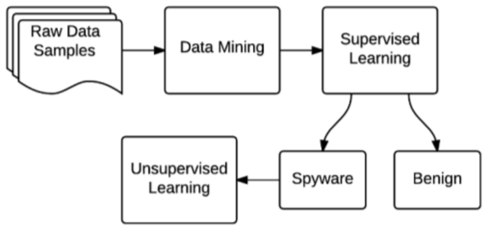


Figure 1: A graphical representation of our experimental procedure

3.1 Sample Collection

Initially, we collected 300 benign files and 120 files containing malware. The benign files were downloaded from two web sources, Download.com and Softpedia.com. These websites both certify that their files are malware free. The benign files we downloaded contain a wide variety of software, including business applications, games, web browsers, and developer tools. The overall size of our benign sample collection is 5.05 GB.

The malicious files were downloaded through collections from VXHeaven and KernelMode.info, both are online communities focusing on malware reversal, development, and research. The malware in the collections are categorized by their behaviors. We primarily downloaded malware that were categorized under Backdoor, Rootkit, Trojan, and Worm. The overall size of our malicious sample collection is 1.28 GB.

3.2 Data Mining/Feature Extraction

With the samples we collected, we now data mine all of the files for features. A feature is an identifying characteristic of the file, and we extract two general types of features from each file: N-grams and Portable Executable structure.

3.2.1 N-grams

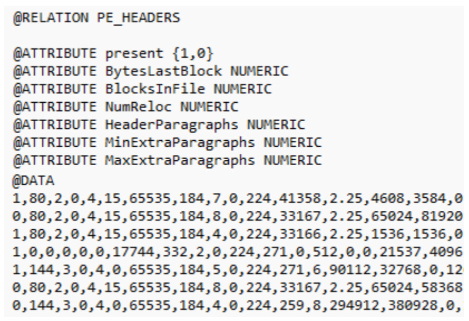
N-grams are sequences of n bytes extracted from a sample's machine code. N-grams are the most naive form of features from a data mining perspective, and can be viewed as DNA strands for virtual files. We first generate a hexadecimal dump - the raw computer data as seen in memory - from each sample. After the hexadecimal dumps are obtained, we select a size of n to sequence the dump into n-grams. Past research from [5] [8] has shown that n-grams of size 5 (each sequence is exactly 5 bytes) produced the most overall accurate results. The dumping and sequencing is performed using a Linux utility called "xxd" and can be used on any Linux distribution.

3.2.2 Portable Executable Features

The Portable Executable structure contains information about how the operating system manages a resources allocated to a program [4]. Features from the PE Header, MZ Header, and Data-Directory were extracted and compared from each sample's structure. In the results section, we compare the performance of our algorithms using selected features from each section of the PE structure. In addition, we extracted the DLL imports and API function calls of each file from the Import Address Table [9]. The PE feature mining was done by through a package from Ruby called "pedump".

3.3 Supervised Machine Learning

The goal of the supervised machine learning implementation is to categorize file inputs as strictly benign or malware. We use the Waikato Environment for Knowledge Analysis (WEKA) as our platform for running supervised algorithms and analyzing performance evaluations. WEKA uses a unique file format as input, an Attribute-Relation File Format (ARFF) database. The format allows us to list all of our features and their type (numeric, nominal, string, etc.), an example can be seen in figure 1. ARFF files for n-grams consist of two attribute fields: the n-gram sequence (numeric) and a "present" field, which indicates if an n-gram is found in malware traces or not (nominal). ARFF files for PE features also contain a present attribute field (nominal), all of the PE fields (numeric), and API function calls (nominal). The ARFF file generation is done through a parser that we created in C++.



```
@RELATION PE_HEADERS

@ATTRIBUTE present {1,0}
@ATTRIBUTE BytesLastBlock NUMERIC
@ATTRIBUTE BlocksInFile NUMERIC
@ATTRIBUTE NumReloc NUMERIC
@ATTRIBUTE HeaderParagraphs NUMERIC
@ATTRIBUTE MinExtraParagraphs NUMERIC
@ATTRIBUTE MaxExtraParagraphs NUMERIC
@DATA
1,80,2,0,4,15,65535,184,7,0,224,41358,2.25,4608,3584,0
0,80,2,0,4,15,65535,184,8,0,224,33167,2.25,65024,81920
1,80,2,0,4,15,65535,184,4,0,224,33166,2.25,1536,1536,0
1,0,0,0,0,0,17744,332,2,0,224,271,0,512,0,0,21537,4096
1,144,3,0,4,0,65535,184,5,0,224,271,6,90112,32768,0,12
0,80,2,0,4,15,65535,184,8,0,224,33167,2.25,65024,58368
0,144,3,0,4,0,65535,184,4,0,224,259,8,294912,380928,0,
```

Figure 2: An example ARFF database generated for a sample's PE features

For n-grams, we first dump all benign and malware files into separate databases. The two databases are then merged into a new database with duplicate n-grams removed, which we will call the common database. N-grams from the common database is compared with the n-grams in the malware database, and if the n-gram is present in the malware database, we indicate so by assigning the n-gram's present field in the ARFF file with a "1."

All of the PE features are numeric, so their values and directly saved into the ARFF database. Due to the large quantity of API function calls, we selected the top 100 most frequent calls among malware and used them as nominal fields in the ARFF database. If a sample performs the specified function call, we assign a "1" to that field and a "0" otherwise.

After the ARFF files have been prepared, we use them as input to our WEKA simulator. WEKA uses a method known as k-fold cross validation to test the supervised algorithms, where the entire data set is partitioned into k sections and one section is used as testing data while the rest are used as the training data. Through this process WEKA is able to compute the true positive rate, false positive rate, and other statistic quantifiers of the algorithm. The three supervised algorithms that we investigated are:

3.3.1 J48 Algorithm

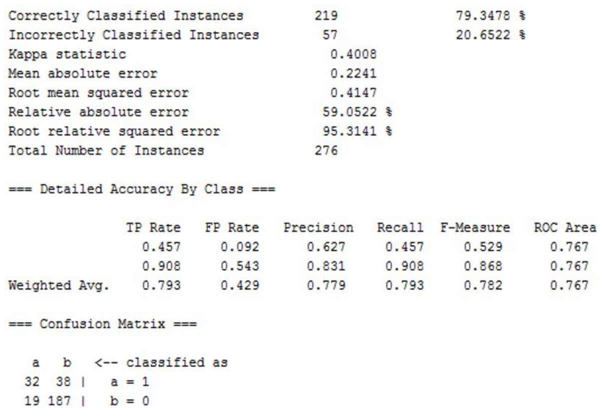
An implementation of the C4.5 decision tree algorithm in Java. This algorithm calculates the normalized information gain ratio from splitting on a feature for all features. The decision tree will create a node that splits on the feature with the highest information gain.

3.3.2 Random Forest

The Random Forest algorithm is an ensemble learning method. It creates several decision trees from the random parts of the data with replacement. The combination of these trees, which contain an approximate of the underlying data, creates a "forest" which gives a much more accurate approximation of the data.

3.3.3 Naïve Bayes Classifier

The Naïve Bayes classifier assumes that all features are independent of each other, and then applies Bayes' Theorem to calculate which node the decision tree will split on.



```
Correctly Classified Instances         219               79.3478 %
Incorrectly Classified Instances        57               20.6522 %
Kappa statistic                          0.4008
Mean absolute error                      0.2241
Root mean squared error                  0.4147
Relative absolute error                 59.0522 %
Root relative squared error             95.3141 %
Total Number of Instances              276

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area
                 0.457     0.092      0.627     0.457     0.529      0.767
                 0.908     0.543      0.831     0.908     0.868      0.767
Weighted Avg.    0.793     0.429      0.779     0.793     0.782      0.767

=== Confusion Matrix ===

   a   b   <-- classified as
  32  38 |   a = 1
  19 187 |   b = 0
```

Figure 3. Output information from WEKA after running a supervised machine learning algorithm on an input sample

3.4 Unsupervised Machine Learning

After the supervised learning algorithms label the samples as malware and benign, the samples classified as malware are then used as input for the unsupervised machine learning. The goals of using unsupervised learning after the supervised algorithm are to classify malware on a severity spectrum and to act as a second buffer to catch any additional false positives. The primary algorithm that was used for unsupervised machine learning was the Self Organizing Feature Map (SOFM).

The SOFM is an artificial neural network that uses neurons that respond to input [1]. The algorithm takes in a data file that is organized similarly to the ARFF format used by WEKA, and outputs a grid of colors where each circle is a neuron.

In order to classify the malware based on their behavior, the features used as input into the SOFM needed to be representative of the functionality of a program. As a result, API function calls, which are found in the DLL imports of the PE structure were used, as they provided the best insight into the behavior of a sample. Our malware samples contained a total of 172,641 different API calls.

Each class of malware has different API function calls which are needed to perform their malicious activity. In our collection of malicious files, there were 4 different classes of malware, including: Backdoor, Rootkit, Trojan, and Worm. In each class, we recorded the top 100 most frequent API calls, ending up with a database of the top identifying API calls of each type of malware.

Using our database of the most frequent API calls of each class, we created a feature vector of 1's and 0's (1 if a certain API call was present, 0 if it was not) for each sample that was labeled "malware" by the supervised learning.

The SOFM we were working with used colors to represent different clusters. The colors are represented numerically with 3 inputs, Red, Green and Blue. To reduce our feature vector to just 3 numbers, we split the vector into groups of 3, and converted the binary sequence into a decimal number. The purpose of this was to give us a one-to-one mapping of the different binary sequences to a unique decimal value. We then took the decimal values and normalized them into a value between 0 and 1, which was finally used as input for the SOFM.

4 Results and Analysis

4.1 Performance Metrics

The performance of each algorithm was compared through the following evaluation metrics:

True Positives (TP): The number of malware correctly identified as malware

False Positives (FP): The number of malware incorrectly identified as benign

True Negatives (TN): The number of benign correctly identified as benign

False Negatives (FN): The number of benign incorrectly identified as malware

True Positive Rate (TPR): Also known as sensitivity,

$$TPR = \frac{TP}{TP + FN}$$

True Negative Rate (TNR): Also known as specificity,

$$TNR = \frac{TN}{TN + FP}$$

Overall Accuracy (ACC):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

In addition to these metrics, we also use a Receiver Operating Characteristics (ROC) curve, which is used in many applications outside of Computer Science to evaluate detection performance. The ROC is a comparison graph of the False Positive Rate (1 - specificity) vs. True Positive Rate (sensitivity). In addition to the curve itself, the *area under the ROC curve* (AUC) can also be used as an evaluation metric in comparison to the overall accuracy. An ideal ROC curve is a unit-step function, where 100% TPR corresponds to 0% FPR and the AUC is exactly 1.0.

4.2 Parameters

For the supervised learning algorithms, we compared the results of each algorithm under different parameters. These included: malicious file percentage (MFP), combinations of different features, feature extraction methods.

It was found that changing the malicious file percentage (MFP) in the training data had a moderate impact on our results. As we increased the number of either benign or malicious files significantly over the other, a negative impact was seen on the performance of our solution. This was due to the fact that with huge imbalances in our sample, the algorithm would generalize files more often, as it was trained with more of one type the other. However, we also found that an exact split in the number of files did not work as well as a slightly skewed split.

Another parameter we changed were the different combinations of features used as input for WEKA. Within the Portable Executable structure, there are many sections of data that can be interpreted. These include the MZ header, PE File Header, Data Directory, and DLL imports (contain API function calls) for a total of 42 unique features. The top 100 most frequent API calls were also recorded, for a total of 142 features. Individually, these sections performed worse than when used in combination with each other. It is interesting to note that when using the MZ header features by themselves, the result was entirely meaningless as it provided an AUC of 0.5.

For feature extraction, we used both information gain and information gain ratio to select our attributes. These extraction methods reduce the amount of "noise" created by useless features that do not tell us anything regarding whether the program is malicious or benign. As a result, all features contained within the MZ header were removed. The number of features were reduced from 142 to 73. In most cases, applying information gain was the most ideal, as it lowered the FPR while the TPR remained relatively the same. On the other hand, using information gain ratio lowered the FPR approximately 2-3 percent, however the TPR suffered a significant reduction. As a result, we determined that information gain was the better feature selection as it maximized the distance between the TPR and the FPR, as well as having a slightly higher AUC.

A comparison of the algorithms reveals that the Random Forest algorithm is the best on average in every metric. It consistently outperformed both the J48 algorithm and the NBTree algorithm, most noticeably in regards to the AUC. The average AUC of Random Forest was 0.9467, while both J48 and NBTree AUC's falling under 0.9. With higher AUC values, the Random Forest algorithm maximized the difference between the TPR and FPR, thus producing the best results.

4.3 Comparison and Optimal Results

As mentioned earlier, the PE structure contains several categories of features in different locations of the structure. The three categories we looked at were the PE header (PE), Data-Directory (DD), and MZ header (MZ). In addition, we extracted the top 100 API function calls for a given set of samples and used them as features. Table 1 is a comparison of our top performing solutions. We found that the Random Forest algorithm almost always produced better results when run under the same conditions as the other algorithms. Also, it was observed that when only the MZ header features are used, all supervised learning algorithms returned inconclusive results.

Table 1. Performance Comparison of Varying Parameters

Features	Algorithm	ACC	TPR	FPR	AUC
PE + DD + API	Random Forest	93.5%	90%	5%	0.960
PE + DD + MZ + API	Random Forest	93%	90%	5.7%	0.959
DD + API	Random Forest	91%	86.7%	7.1%	0.963
PE + API	J48	90.5%	85%	7.1%	0.875
PE + DD + MZ + API	J48	90%	81.7%	6.4%	0.895
MZ + PE + API	Naïve Bayes	91%	88.3%	7.9%	0.912
API	Naïve Bayes	89%	88.3%	10.7%	0.918

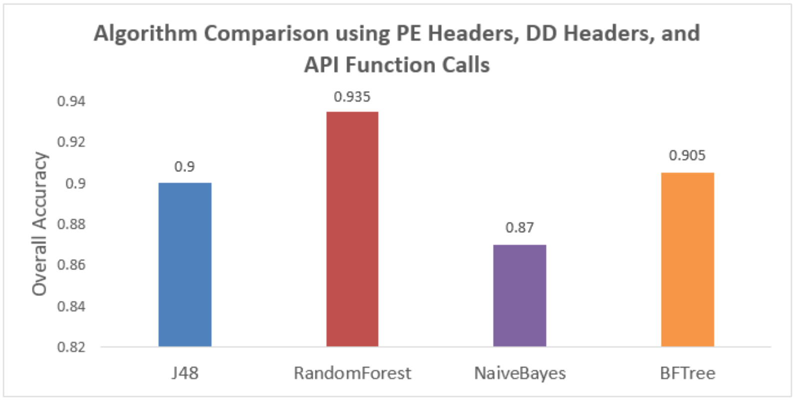


Figure 4. Comparison of algorithms' overall accuracy using the same parameters

In figure 4, we compare the overall accuracy of each algorithm using the same parameters. The features used were PE headers, DD headers, and API function calls. We saw the highest performing algorithm was RandomForest, which we use to further test the effectiveness of feature selection.

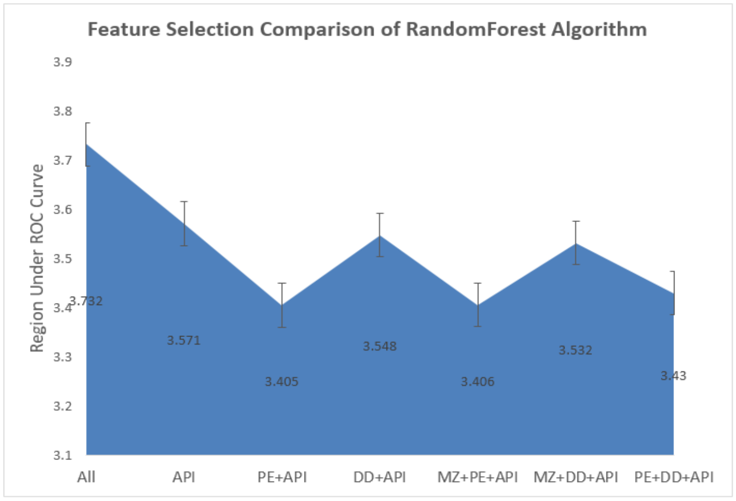


Figure 5. A comparison of area under the ROC curve using different features with the Random Forest algorithm

After running multiple trials, using different combinations of each parameter, we achieved our best result. This was done with the following parameters: 70% benign files to 30% malicious files, done under the Random Forest Algorithm, using the PE Header in conjunction with the Data Directory and top 100 most frequent API calls, all applied with information gain. The result was a feature vector containing 73 identifying features for malware. **This gave us an overall accuracy of 94%, a TPR of 91.7%, an FPR of 5% and an AUC of 0.962.** While we did achieve higher TPR and lower FPR, we defined the "best" result as the trial that maximized the difference between the TPR and the FPR.

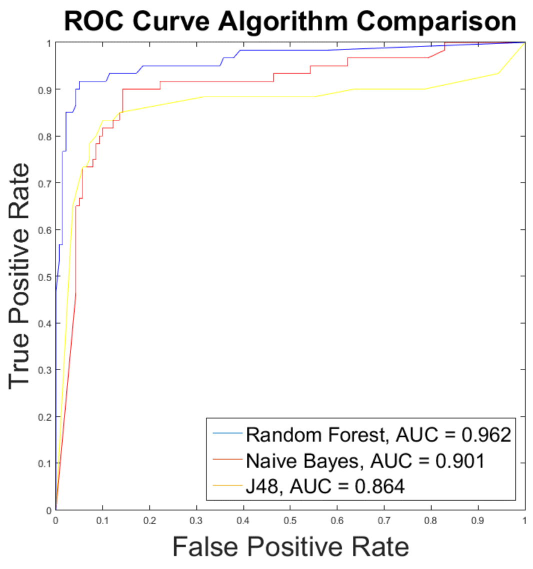


Figure 6. The ROC curve comparison for the top performing parameters of each supervised algorithm. Our overall best solution is shown in blue.

4.4 Self-Organizing Feature Map Classification

The SOFM algorithm we implemented represents neurons as colors by having three fields for RGB values ranging between 0 and 1. Our input data, the API function calls, need to be reduced to three fields and then normalized. The normalized value of each input is calculated by:

$$X_{i,norm} = \frac{X_i - X_{min}}{X_{max} - X_{min}}$$

In addition, we calculated the 6-norm distance from each input to the neuron rather than the Euclidean distance used in general SOFMs. The equation to calculate the p-norm distance from input to neuron is as shown below. Using the 6-norm provided a clearer output spectrum and each category of malware can be easily seen.

$$L_p = \left(\sum_{i=1}^{n} |q_i - p_i|^p \right)^{1/p}$$

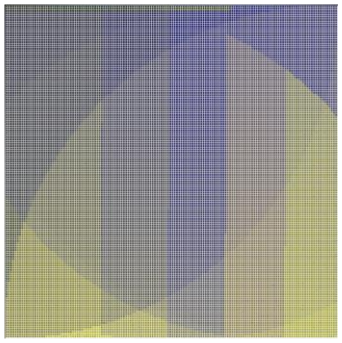
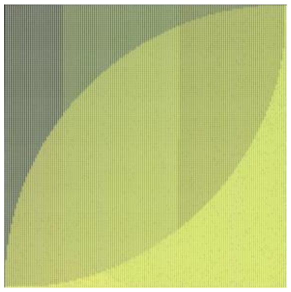


Figure 7. Snapshot of the SOFM in the process of clustering related malware into a severity spectrum

For an epoch of 3000 iterations and an initial learning rate of 0.3, were able to generate the severity spectrums shown in Figure 4 and 5. For the sake of clarity on the color clusters, we used a total of 22,500 (150x15) neurons in the network.

Figure 8. A completed SOFM output severity spectrum after 3000 iterations, each category of malware can be distinguished from the different colors of neurons



5 Conclusion and Future Work

In our investigation, we introduced a new method of hybrid machine learning to detect spyware and classify spyware on a severity spectrum. We explored the variations of parameters and their effect on the accuracy of detection. It was discovered that the best performing solution uses features mined from the PE structure and API function calls selected through information gain applied with the Random Forest algorithm. We achieved a 94% overall accuracy with 91.7% TPR and 5% FPR.

For future work, we suggest the use of more features in conjunction with each other. Our research combined the use of PE features with API function calls, using information gain to reduce our features. Additional features that could be examined are: opcode n-grams, and string analysis to find keywords representing maliciousness. The combination of these features used with the correct feature extraction would minimize the counts of false positives.

For the unsupervised machine learning, more algorithms in clustering and neural networks can be explored. Algorithms such as K-means clustering and hierarchical clustering could provide more accurate groupings of malware. In addition, a non-color based representation of the SOFM can be used. Disregarding color will give additional depth and dimensions to the SOFM, allowing it to cluster samples in more than just a two dimensional grid.

6 ACKNOWLEDGEMENTS

This research work was conducted at Oakland University in the UnCoRe program - REU site supported by the National Science Foundation under Grant No. CNS-1460897. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The authors would like to thank Dr. Huirong Fu, PI and Program Coordinator of the UnCoRe REU program at Oakland University, for her support and guidance.

REFERENCES

[1]. Abdel-Aty-Zohdy, H. S., & Zohdy, M. A. (2001). Self-organizing feature maps. Wiley encyclopedia of electrical and electronics engineering () John Wiley & Sons, Inc. doi:10.1002/047134608X.W5114

[2]. Akpojaro, J., Aigbe, P., Onwudebelu,U. (2014). Unsupervised machine learning techniques for detecting malware applications in wireless devices. Transactions on Machine Learning and Artificial Intelligence, 2(3)

[3]. Bahraminikoo, P., Samiei yeganeh, M., & Babu, G. P. (2012). Utilization data mining to detect spyware. IOSR Journal of Computer Engineering, 4(3)

[4]. Baldangombo, U., Jambaljav, N., & Horng, S. (2013). A static malware detection system using data mining methods. Corr, abs/1308.2831

[5]. Chavan, M. k., & Zende, D. A. (2013). Spyware solution: Detection of spyware by data mining and machine learning technique. International Conference on Advanced Research in Engineering and Technology, Vijayawada, India.

[6]. Elovici, Y., Shabtai, A., Moskovitch, R., Tahan, G., & Glezer, C. (2007). Applying machine learning techniques for detection of malicious code in network traffic.4667, 44-50. doi:10.1007/978-3-540-74565-5_5

[7]. Lavesson, N., Boldt, M., Davidsson, P., & Jacobsson, A. (2011). Learning to detect spyware using end user license agreements. Knowledge and Information Systems, 26(2), 285-307.

[8]. Shahzad, R. K., Haider, S. I., & Lavesson, N. (2010). Detection of spyware by mining executable files. Availability, Reliability, and Security, 2010. ARES '10 International Conference on, Krakow. 295-302.

[9]. Singhal, P., & Raul, N.Malware detection module using machine learning algorithms to assistin centralized security in enterprise networks.

[10]. Wang, T., Horng, S., Su, M., Wu, C., Wang, P., & Su, W. (2006). A surveillance spyware detection system based on data mining methods. IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. 3236-3241.