Evolving Computational Intelligence System for Malware Detection

Konstantinos Demertzis and Lazaros Iliadis

Democritus University of Thrace, Department of Forestry & Management of the Environment & Natural Resources, 193 Pandazidou st., 68200 N Orestiada, Greece
kdemertz@fmenr.duth.gr, liliadis@fmenr.duth.gr

Abstract. Recent malware developments have the ability to remain hidden during infection and operation. They prevent analysis and removal, using various techniques, namely: obscure filenames, modification of file attributes, or operation under the pretense of legitimate programs and services. Also, the malware might attempt to subvert modern detection software, by hiding running processes, network connections and strings with malicious URLs or registry keys. The malware can go a step further and obfuscate the entire file with a packer, which is special software that takes the original malware file and compresses it, thus making all the original code and data unreadable. This paper proposes a novel approach, which uses minimum computational power and resources, to indentify Packed Executable (PEX), so as to spot the existence of malware software. It is an Evolving Computational Intelligence System for Malware Detection (ECISMD) which performs classification by Evolving Spiking Neural Networks (eSNN), in order to properly label a packed executable. On the other hand, it uses an Evolving Classification Function (ECF) for the detection of malwares and applies Genetic Algorithms to achieve ECF Optimization.

Keywords: Security, Packed Executable, Malware, Evolving Spiking Neural Networks, Evolving Classification Function, Genetic Algorithm for Offline ECF Optimization.

1 Introduction

Malware is a kind of software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems. It can appear in the form of code, scripts, active content, or any other. To identify already known malware, existing commercial security applications search a computer's binary files for predefined signatures. However, obfuscated viruses use software packers to protect their internal code and data structures from detection. Antivirus scanners act like file filters, inspecting suspicious file loading and storing activities. Malicious programs with obfuscated content, can bypass antivirus scanners. Eventually, they are unpacked and executed in the victim's system [1].

L. Iliadis, M. Papazoglou, and K. Pohl (Eds.): CAiSE 2014 Workshops, LNBIP 178, pp. 322–334, 2014.

Code packing is the dominant technique used to obfuscate malicious code, to hinder an analyst's understanding of the malware's intent and to evade detection by Antivirus systems. Malware developers, transform executable code into data, at a post-processing stage in the whole implementation cycle. This transformation uses static analysis and it may perform compression or encryption, hindering an analyst's understanding. At runtime, the data or hidden code is restored to its original executable form, through dynamic code generation using an associated restoration routine. Execution then resumes as normal to the original entry point, which marks the entry point of the original malware, before the code packing transformation is applied. Finally, execution becomes transparent, as both code packing and restoration have been performed. After the restoration of one packing, control may transfer another packed layer. The original entry point is derived from the last such layer [2].

Code packing provides compression and software protection of the intellectual properties contained in a program. It is not necessarily advantageous to flag all occurrences of code packing as indicative of malicious activity. It is advisable to determine if the packed contents are malicious, rather than identifying only the fact that unknown contents are packed. Unpacking is the process of stripping the packer layers off packed executables to restore the original contents in order to inspect and analyze the original executable signatures. Universal unpackers, introduce a high computational overhead, low convergence speed and computational resource requirements. The processing time may vary from tens of seconds to several minutes per executable. This hinders virus detection significantly, since without a priori knowledge on the nature of the executables to be checked for malicious code all of them would need to be run through the unpacker. Scanning large collections of executables, may take hours or days. This research effort aims in the development and application of an innovative, fast and accurate Evolving Computational Intelligence System for Malware Detection (ECISMD) approach for the identification of packed executables and detection of malware by employing eSNN. A multilayer ECF model has been employed for malware detection, which is based on fuzzy clustering. Finally, an evolutionary Genetic Algorithm (GA) has been applied to optimize the ECF network and to perform feature extraction on the training and testing datasets. A main advantage of ECISMD is the fact that it reduces overhead and overall analysis time, by classifying packed or not packed executables.

1.1 Literature Review

Dynamic unpacking approaches monitor the execution of a binary in order to extract its actual code. These methods execute the samples inside an isolated environment that can be deployed as a virtual machine or an emulator [3]. The execution is traced and stopped when certain events occur. Several dynamic unpackers use heuristics to determine the exact point where the execution jumps from the unpacking routine to the original code. Once this point is reached, the memory content is bulk to obtain an unpacked version of the malicious code. Other approaches for generic dynamic unpacking have been proposed that are not highly based on heuristics such as PolyUnpack [4] Renovo [5], OmniUnpack [6] or Eureka [7]. However, these methods are very tedious and time consuming, and cannot counter conditional execution of unpacking routines, a technique used for anti-debugging and anti-monitoring defense [8]. Another common approach is

using the structural information of the executables to train supervised machine-learning classifiers to determine if the sample under analysis is packed or if it is suspicious of containing malicious code (e.g., PEMiner [9], PE-Probe [10] and Perdisci et al. [11]). These approaches that use this method for filtering, previous to dynamic unpacking, are computationally more expensive and time consuming and less effective to analyze large sets of mixed malicious and benign executables [12] [13] [14].

Artificial Intelligence and data mining algorithms have been applied as malicious detection methods and for the discovery of new malware patterns [15]. In the research effort of Babar and Khalid [3], boosted decision trees working on n-grams are found to produce better results than Naive Bayes classifiers and Support Vector Machines (SVM). Ye et al., [16] use automatic extraction of association rules on Windows API execution sequences to distinguish between malware and clean program files. Chandrasekaran et al., [17] are using association rules, on honeytokens of known parameters. Chouchan et al., [18] used Hidden Markov Models to detect whether a given program file is (or is not) a variant of a previous program file. Stamp et al., [19] employ profile hidden Markov Models, which have been previously used for sequence analysis in bioinformatics. The capacity of neural networks (ANN) to detect polymorphic malware is explored in [20]. Yoo [21] employs Self-Organizing Maps to identify patterns of behavior for viruses in Windows executable files. These methods they have low accuracy as a consequence, packed benign executables would likely cause false alarm, whereas malware may remain undetected.

2 Methodologies Comprising the Proposed Hybrid Approach

2.1 Evolving Spiking Neural Networks (eSNN)

eSNN are modular connectionist-based systems that evolve their structure and functionality in a continuous, self-organized, on-line, adaptive, interactive way from incoming information. These models use trains of spikes as internal information representation rather than continuous variables [22]. The eSNN developed and discussed herein is based in the “Thorpe” neural model [23]. This model intensifies the importance of the spikes taking place in an earlier moment, whereas the neural plasticity is used to monitor the learning algorithm by using one-pass learning. In order to classify real-valued data sets, each data sample, is mapped into a sequence of spikes using the Rank Order Population Encoding (ROPE) technique [24] [25]. The topology of the developed eSNN is strictly feed-forward, organized in several layers and weight modification occurs on the connections between the neurons of the existing layers.

The ROPE method is alternative to the conventional rate coding scheme (CRCS). It uses the order of firing neuron’s inputs to encode information. This allows the mapping of vectors of real-valued elements into a sequence of spikes. Neurons are organized into neuronal maps which share the same synaptic weights. Whenever the synaptic weight of a neuron is modified, the same modification is applied to the entire population of neurons within the map. Inhibition is also present between each neuronal map. If a neuron spikes, it inhibits all the neurons in the other maps with neighboring positions.

This prevents all the neurons from learning the same pattern. When propagating new information, neuronal activity is initially reset to zero. Then, as the propagation goes on, each time one of their inputs fire, neurons are progressively desensitize. This is making neuronal responses dependent upon the relative order of firing of the neuron's afferents [24], [26], [27].

The aim of the one-pass learning method is to create a repository of trained output neurons during the presentation of training samples. After presenting a certain input sample to the network, the corresponding spike train is propagated through the eSNN which may result in the firing of certain output neurons. It is possible that no output neuron is activated and the network remains silent and the classification result is undetermined. If one or more output neurons have emitted a spike, the neuron with the shortest response time among all activated output neurons is determined. The label of this neuron is the classification result for the presented input [26], [27], [28].

2.2 Evolving Connectionist Systems (ECOS)

ECOS [29] are multi-modular, connectionist architectures that facilitate modeling of evolving processes and knowledge discovery [26]. An ECOS is an ANN operating continuously in time and adapting its structure and functionality through a continuous interaction with the environment and other systems according to: (i) a set of parameters that are subject to change; (ii) an incoming continuous flow of information with unknown distribution; (iii) a goal (rational) criterion (subject to modification) applied to optimize the performance of the system. The evolving connectionist systems evolve in an open space, using constructive processes, not necessarily of fixed dimensions. They learn in on-line incremental fast mode, possibly through one pass of data propagation. Life-long learning is a main attribute of this procedure. They operate as both individual systems, and as part of an evolutionary population of such systems. [26] [30]. ECOS are connectionist structures that evolve their nodes and connections through supervised incremental learning from input-output data.

Their architecture comprises of five layers: input nodes, representing input variables; input fuzzy membership nodes, representing the membership degrees of the input values to each of the defined membership functions; rule nodes, representing cluster centers of samples in the problem space and their associated output function; output fuzzy membership nodes, representing the membership degrees to which the output values belong to defined membership functions; and output nodes, representing output variables [31].

ECOS learn local models from data through clustering of the data and associating a local output function for each cluster. Rule nodes evolve from the input data stream to cluster the data, and the first layer W_1 connection weights of these nodes represent the coordinates of the nodes in the input space. The second layer W_2 represents the local models (functions) allocated to each of the clusters. Clusters of data are created based on similarity between data samples either in the input space, or in both the input space and the output space. Samples that have a distance to an existing cluster center (rule node) N of less than a threshold R_{max} are allocated to the same cluster N_c. Samples that do not fit into existing clusters, form new clusters as they arrive in time. Cluster centers are continuously adjusted according to new data samples and new clusters are created incrementally. The similarity between a sample S = (x, y) and an existing rule

node N = (W_1, W_2) can be measured in different ways, the most popular of them being the *normalized Euclidean distance* given by equation 1, where *n* is the number of the input variables. $d(S,N) = \frac{1}{n}\left[\sum_{i=1}^{n} |x_i - W_{1N}|^2\right]^{\frac{1}{2}}$ (1).

ECOS learn from data and automatically create a local output function for each cluster, the function being represented in the W_2 connection weights, creating local models. Each model is represented as a local rule with an antecedent –the cluster area, and a consequent– the output function applied to data in this cluster.

2.3 Evolving Classification Function and Genetic Algorithms

ECF, a special case of ECOS used for pattern classification, generates rule nodes in an N dimensional input space and associate them with classes. Each rule node is defined with its centre, radius (influence field) and the class it belongs to. A learning mechanism is designed in such a way that the nodes can be generated. The ECF model used here is a connectionist system for classification tasks that consists of four layers of neurons (nodes). The first layer represents the input variables; the second layer – the fuzzy membership functions; the third layer represents clusters centers (prototypes) of data in the input space; and the four layer represents classes [30], [26]. A GA is an evolutionary algorithm in which the principles of the Darwin's theory are applied to a population of solutions in order to "breed" better solutions. Solutions, in this case the parameters of the ECF network, are encoded in a binary string and each solution is given a score depending on how well it performs. Good solutions are selected more frequently for breeding and are subjected to crossover and mutation. After several generations, the population of solutions should converge on a "good" solution. The ECF model and the GA algorithm for Offline ECF Optimization are parts from NeuCom software (*http://www.kedri.aut.ac.nz/*) which is a Neuro-Computing Decision Support Environment, based on the theory of ECOS [29].

3 Description of the Proposed Hybrid ECISMD Algorithm

The proposed herein, hybrid ECISMD methodology uses an eSNN classification approach to classify packed or unpacked executables with minimum computational power combined with the ECF method in order to detect packed malware. Finally it applies Genetic Algorithm for ECF Optimization, in order to decrease the level of false positive and false negative rates. The general algorithm is described below:

Step 1: The train and test *datasets* are determined and formed, related to *n* features. The required classes (packed and unpacked executables) that use the variable *Population Encoding* are imported. This variable controls the conversion of real-valued data samples into the corresponding time spikes. The encoding is performed with 20 Gaussian receptive fields per variable (Gaussian width parameter beta=1.5). The data are normalized to the interval [-1,1] and so the coverage of the Gaussians is determined by using i_min and i_max. For the normalization processing the following function 2 is used:

$$x_{1_{norm}} = 2 * \left(\frac{x_1 - x_{min}}{x_{max} - x_{min}}\right) - 1, \quad x \in R \tag{2}$$

The data is classified in two classes namely: ***Class 0*** which contains the unpacked results and ***Class 1*** which comprises of the packed ones. The eSNN is using modulation factor m=0.9, firing threshold ratio c=0.7 and similarity threshold s=0.6 in agreement with the vQEA algorithm [28] [27]. More precisely, let A = { a_1 , a_2 , a_3 ... a_{m-1} , a_m } be the ensemble of afferent neurons of neuron i and W = {$w_{1,i}$, $w_{2,i}$, $w_{3,i}$ ... $w_{m-1,i}$, $w_{m,i}$} the weights of the m corresponding connections; let mod ∈ [0,1] be an arbitrary modulation factor. The activation level of neuron i at time t is given by equation 3: $\text{Activation}(i,t)=\sum_{j\in[1,m]} \text{mod}^{\text{order}(a_j)} w_{j,i}$ (3) where order(a_j) is the firing rank of neuron a_j in the ensemble A.

By convention, order(a_j)=+8 if a neuron a_j is not fired at time t, sets the corresponding term in the above sum to zero. This kind of desensitization function could correspond to a fast shunting inhibition mechanism. When a neuron reaches its threshold, it spikes and inhibits neurons at equivalent positions in the other maps so that only one neuron will respond at any location. Every spike triggers a time based Hebbian-like learning rule that adjusts the synaptic weights. Let t_e be the date of arrival of the Excitatory PostSynaptic Potential (EPSP) at synapse of weight W and t_a the date of discharge of the postsynaptic neuron.

If $t_{e<}$ t_a then $dW=a(1-W)e^{-|\Delta o|\tau}$ else $dW=-aW\, e^{-|\Delta o|\tau}$ (4). Δo is the difference between the date of the EPSP and the date of the neuronal discharge (expressed in term of order of arrival instead of time), a is a constant that controls the amount of synaptic potentiation and depression [24]. ROPE technique with receptive fields, allow the encoding of continuous values [26] [27]. Each input variable is encoded independently by a group of one-dimensional receptive fields (figure 1). For a variable n, an interval $[I^n_{min,}\ I^n_{max}]$ is defined. The Gaussian receptive field of neuron i is given by its center μi and width σ by equation 6.: $\mu i = I^n_{min} + \frac{2i-3}{2}\frac{I^n_{max} - I^n_{min}}{M-2}$ (5)

$\sigma=\frac{1}{\beta}\frac{I^n_{max} - I^n_{min}}{M-2}$ (6) where $1\leq\beta\leq2$ and the parameter β directly controls the width of each Gaussian receptive field. Figure 1 depicts an encoding example of a single variable.

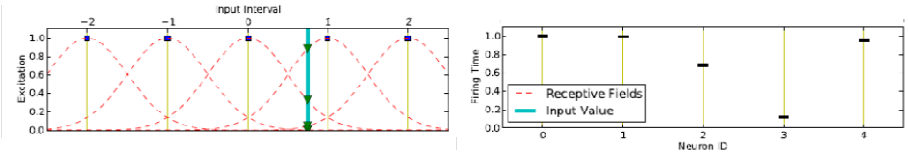

Fig. 1. Population encoding based on Gaussian receptive fields. Left Figure: Input Interval – Right Figure: Neuron ID

For an input value v=0.75 (thick straight line) [27] the intersection points with each Gaussian is computed (triangles), which are in turn translated into spike time delays (right figure) [27].

Step 2: The eSNN is trained with the *packed_train* dataset vectors and the testing is performed with the *packed_test* vectors. The procedure of one pass learning is described in the following Algorithm1 [26] [27].

Algorithm 1: Training an evolving Spiking Neural Network (eSNN) [27]

Require: m_l, s_l, c_l **for a class label** l ∈ L
1: initialize neuron repository $R_l = \{\}$
2: **for all** samples $X^{(i)}$ belonging to class l **do**
3: $w_j^{(i)} \leftarrow (m_l)^{order(j)}$, ∀ j | j pre-synaptic neuron of i
4: $u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)}$
5: $\theta^{(i)} \leftarrow c_l u_{max}^{(i)}$
6: **if** $\min(d(w^{(i)}, w^{(k)})) < s_l$, $w^{(k)} \in R_l$ **then**
7: $w^{(k)} \leftarrow$ merge $w^{(i)}$ and $w^{(k)}$ according to Equation 7
8: $\theta^{(k)} \leftarrow$ merge $\theta^{(i)}$ and $\theta^{(k)}$ according to Equation 8
9: **else**
10: $R_l \leftarrow R_l \cup \{w^{(i)}\}$
11: **end if**
12: **end for**

For each training sample i with class label l ∈ L a new output neuron is created and fully connected to the previous layer of neurons resulting in a real-valued weight vector $w^{(i)}$ *with* $w_j^{(i)} \in R$ denoting the connection between the pre-synaptic neuron j and the created neuron i. In the next step, the input spikes are propagated through the network and the value of weight $w_j^{(i)}$ is computed according to the order of spike transmission through a synapse j: $w_j^{(i)} = (m_l)^{order(j)}$, ∀ j | j pre-synaptic neuron of i.

Parameter m_l is the modulation factor of the Thorpe neural model. Differently labeled output neurons may have different modulation factors m_l. Function order(j) represents the rank of the spike emitted by neuron j. The firing threshold $\theta^{(i)}$ of the created neuron I is defined as the fraction $c_l \in R$, $0 < c_l < 1$, of the maximal possible potential

$$u_{max}^{(i)} : \ \theta^{(i)} \leftarrow c_l u_{max}^{(i)} \ (7) \quad u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)} \tag{8}$$

The fraction c_l is a parameter of the model and for each class label l ∈ L a different fraction can be specified. The weight vector of the trained neuron is compared to the weights corresponding to neurons already stored in the repository. Two neurons are considered too "similar" if the minimal *Euclidean* distance between their weight vectors is smaller than a specified similarity threshold s_l (the eSNN object uses optimal similarity threshold s=0.6). All parameters of eSNN (modulation factor m_l, similarity threshold s_l, PSP fraction c_l, l ∈ L) included in this search space, were optimized according to the Versatile Quantum-inspired Evolutionary Algorithm (vQEA) [28]. Both the firing thresholds and the weight vectors were merged according to equations 9 and 10:

$$w_j^{(k)} \leftarrow \frac{w_j^{(i)} + N w_j^{(k)}}{1+N}, \ \forall \ j \mid j \text{ pre-synaptic neuron of i} \tag{9}$$

$$\theta^{(k)} \leftarrow \frac{\theta^{(i)} + N\theta^{(k)}}{1+N} \tag{10}$$

Integer N denotes the number of samples previously used to update neuron k. The merging is implemented as the (running) average of the connection weights, and the (running) average of the two firing thresholds [27]. After merging, the trained neuron i is discarded and the next sample processed. If no other neuron in the repository is similar to the trained neuron i, the neuron i is added to the repository as a new output.

Step 3: If the result is unpacked then the process is terminated and the executable file goes to the antivirus scanner. If the result of the classification is packed, the new classification process is initiated employing the ECF method. This time the malware data vectors are used. These vectors comprise of 9 features and 2 classes malware and benign. The learning algorithm of the ECF according to the ECOS is as follows:

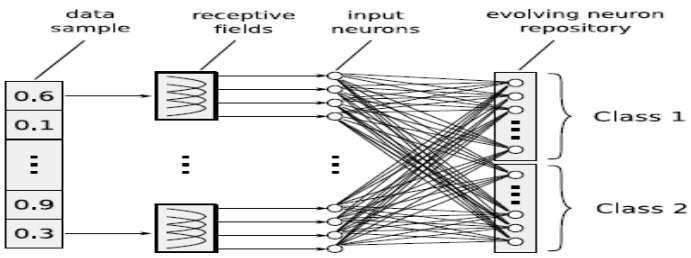

Fig. 2. The Evolving Spiking Neural Network (eSNN) architecture

a. If all input vectors are fed, finish the iteration; otherwise, input a vector from the data set and calculate the distances between the vector and all rule nodes already created using Euclidean distance. **b.** If all distances are greater than a max-radius parameter, a new rule node is created. The position of the new rule node is the same as the current vector in the input data space and the radius of its receptive field is set to the min-radius parameter; the algorithm goes to step 1; otherwise it goes to the next step. **c.** If there is a rule node with a distance to the current input vector less than or equal to its radius and its class is the same as the class of the new vector, nothing will be changed; go to step 1; otherwise. **d.** If there is a rule node with a distance to the input vector less than or equal to its radius and its class is different from those of the input vector, its influence field should be reduced. The radius of the new field is set to the larger value from the two numbers: distance minus the min-radius; min radius. New node is created as in to represent the new data vector. **e.** If there is a rule node with a distance to the input vector less than or equal to the max-radius, and its class is the same as of the input vector's, enlarge the influence field by taking the distance as a new radius if only such enlarged field does not cover any other rule nodes which belong to a different class; otherwise, create a new rule node in the same way as in step 2, and go to step 1 [33].

Step 4: To increase the level of integrity the Offline ECF Optimization with GA is used. ECF system is an ANN that operates continuously in time and adapts its structure and functionality through a continuous interaction with the environment and with other systems. This is done according to a set of parameters P that are subject to change during the system operation; an incoming continuous flow of information with unknown distribution; a goal (rationale) criteria that is applied to optimize the performance. The set of parameters P of an ECOS can be regarded as a chromosome of "*genes*" of the

evolving system and evolutionary computation can be applied for their optimization. The GA algorithm for offline ECF Optimization runs over generations of populations and standard operations are applied such as: binary encoding of the genes (parameters); roulette wheel selection criterion; multi-point crossover operation for crossover. Genes are complex structures and they cause dynamic transformation of one substance into another during the life of an individual, as well as the life of the human population over many generations. Micro-array gene expression data can be used to evolve the ECF with inputs being the expression level of a certain number of selected genes and the outputs being the classes. After the ECF is trained on gene expression rules can be extracted that represent packed or unpacked [34].

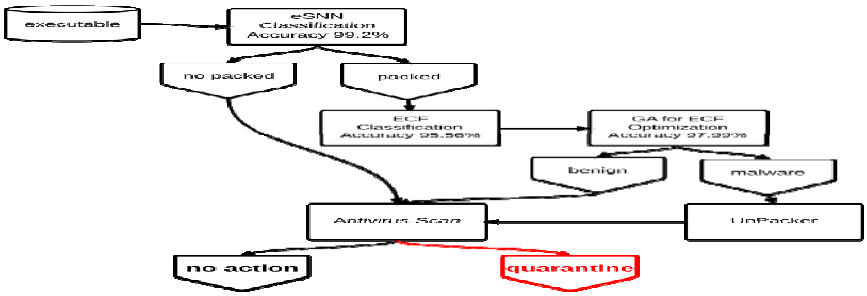

Fig. 3. Graphical display of the ECISMD Algorithm

Step 5: If the result of the classification is benign, the executable file goes to antivirus scanner and the process is terminated. Otherwise, the executable file is marked as malicious, it goes to the unpaker, to the antivirus scanner for verification and finally placed in quarantine and the process is terminated.

4 Data and Results

To prove generalization ability of our classification approach we need a reliable dataset. The full_dataset comprised of 2,598 packed viruses from the Malfease Project dataset (http://malfease.oarci.net), 2,231 non-packed benign executables collected from a clean installation of *Windows XP Home plus*, several common user applications and 669 packed benign executables. It was divided randomly in two parts: 1) a training dataset containing 2,231 patterns related to the non-packed benign executable and 2,262 patterns related to the packed executables detected using unpacked software 2) a testing dataset containing 1,005 patterns related to the packed executables that even the most well know unpacked software was not able to detect. These datasets are available at http://roberto.perdisci.googlepages.com/code [11].

The virus dataset containing 2,598 malware and 669 benign executables is divided in two parts: 1) a training dataset containing 1,834 patterns related to the malware and 453 patterns related to the benign executables 2) a test dataset containing 762 patterns related to the malware and 218 benign executables. In order to translate each

executable into a pattern vector Perdisci et al [11] they use binary static analysis, to extract information such as, the name of the code and data sections, the number of writable-executable sections, the code and data entropy.

In both classifications described below, *Training Accuracy* reports the average accuracy computed over 10-fold cross-validation. *Testing Accuracy* refers to the percentage of packed executables that were correctly detected by each classifier in the *Packed_Test_Dataset* and in the *Virus_Test_Dataset* respectively.

In the first classification performed by the ECISMD, the eSNN approach was employed in order to classify packed or not packed executables. The results for testing are: Classification Accuracy: 99.2% No. of evolved neurons: Class 0/867 neurons - Class 1/734 neurons, In order to perform comparison with different learning algorithms the *Weka software version 3.7* was used (http://www.cs.waikato.ac.nz/ml/weka). Table 1 reports the results obtained with *RBF ANN, Naïve Bayes, Multi Layer Perceptron (MLP), Support Vector Machine (LibSVM), k-Nearest-Neighbors* (k-NN) and *eSNN*.

Table 1. Comparison of various approaches for the Packed dataset

Packed Dataset		
Classifier	**Train Accuracy**	**Test Accuracy**
RBFNetwork	98.3085%	98.0859%
NaiveBayes	98.3975%	97.1144%
MLP	99.5326%	96.2189%
LibSVM	99.4436%	89.8507%
k-NN	99.4436%	96.6169%
eSNN	99.8%	99.2%

In the 2nd classification performed by the ECISMD the ECF approach was employed in order to classify malware or benign executables. The ECF model has the following parameter values: MaxField=1, MinField=0.01, number of fuzzy membership functions MF=1; number of rule nodes used to calculate the output value of the ECF when a new input vector is presented MofN=9 (number of neighbors to consider when evaluating nearest node); number of iterations for presenting each input vector Epochs=6. The results for the *test_dataset* are: Classification Accuracy: 95.561%, Correct Samples: 933/980, Accuracy/Class: 82%/ Class 0 - 98% /Class 1. For the ECF parameter optimization during training, the following parameter value ranges were used: Min Field: 0.1, Max Field: 0.8, membership function: 9, Value for the m-of-n parameter: 3, Generation: 6 and Population: 4. For the optimized value of the ECF, 30% of the data was selected for training and 70% for testing. The classification accuracy in test_dataset after the optimization was 97.992%.

Table 2, reports the results obtained with 6 classifiers and optimized ECF network (RBF Network, Naïve Bayes, MLP, Lib SVM, k-NN, ECF and optimized ECF). The best results on the testing dataset were obtained by using the eSNN classifier, to classify packed or not packed executables and the optimized ECF (in the 2nd classification) which classifies malware or benign executables.

Table 2. Comparison of various approaches for the virus dataset

Virus Dataset		
Classifier	**Train Accuracy**	**Test Accuracy**
RBFNetwork	94.4031%	93.0612%
NaiveBayes	94.0533%	92.3469%
MLP	97.7551%	97.289%
LibSVM	94.6218%	94.2857%
k-NN	98.1198%	96.8367%
ECF	99.05%	95.561%
Optimized ECF	99.87%	97.992%

The average time for the classification of a pattern vector on a 3.06GHz Intel P4 processor was about t = 0.00016 seconds per executable. These results are much smaller than the ones of Perdisci et al [11] in which on a 2GHz Dual Core AMD Opteron processor was about t=0.001 seconds per executable.

5 Discussion - Conclusions

A new Evolving Computational Intelligence System for Malware Detection (ECISMD) was introduced. It performs classification using eSNN to properly label a packed executable and ECF with GA to detect malware and to optimize itself towards better generalization. An effort was made to use minimum computational power and resources. The classification performance of the eSNN method and the accuracy of the ECF model were experimentally explored based on different datasets. The eSNN was applied to an unknown dataset and reported promising results. Moreover the ECF model and the genetically optimized ECF network, detects the patterns and classifies them with high accuracy and adds a higher degree of integrity to the rest of the security infrastructure of ECISMD. As a future direction, aiming to improve the efficiency of biologically realistic ANN for pattern recognition, it would be important to evaluate the eSNN model with ROC analysis and to perform feature minimization in order to achieve minimum processing time. Other coding schemes could be explored and compared on the same security task. Finally, the ECISMD could be improved towards a better online learning with self-modified parameter values.

References

1. Yan, W., Zhang, Z., Ansari, N.: Revealing Packed Malware. IEEE (2007)
2. Cesare, S., Xiang, Y.: Software Similarity and Classification. Springer (2012)
3. Babar, K., Khalid, F.: Generic unpacking techniques. In: Proceedings of the 2nd International Conference on Computer, Control and Communication (IC4), pp. 1–6. IEEE (2009)
4. Royal, P., Halpin, M., Dagon, D., Edmonds, R.: Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: ACSAC, pp. 289–300 (2006)
5. Kang, M., Poosankam, P., Yin, H.: Renovo: A hidden code extractor for packed executables. In: 2007 ACM Workshop on Recurring Malcode, pp. 46–53. ACM (2007)

6. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware. In: Proceedings of the ACSAC, pp. 431–441 (2007)
7. Yegneswaran, V., Saidi, H., Porras, P., Sharif, M.: Eureka: A framework for enabling static analysis on malware, Technical report, Technical Report SRI-CSL-08-01 (2008)
8. Danielescu, A.: Anti-debugging and anti-emulation techniques: Code-Breakers J. (2008)
9. Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 121–141. Springer, Heidelberg (2009)
10. Shaq, M., Tabish, S., Farooq, M.: PE-Probe: Leveraging Packer Detection and Structural Information to Detect Malicious Portable Executables. In: Virus Bulletin Conference (2009)
11. Perdisci, R., Lanzi, A., Lee, W.: McBoost: Boosting scalability in malware collection and analysis using statistical classiffication of executables. In: Proceedings of the 2008 Annual Computer Security Applications Conference, pp. 301–310 (2008) ISSN 1063-9527
12. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research 7, 2721–2744 (2006)
13. Ugarte-Pedrero, X., Santos, I., Bringas, P.G., Gastesi, M., Esparza, J.M.: Semi-supervised Learning for Packed Executable Detection. IEEE (2011) 978-1-4577-0460-4/11
14. Ugarte-Pedrero, X., Santos, I., Laorden, C., Sanz, B., Bringas, G.P.: Collective Classification for Packed Executable Identification. In: ACM CEAS, pp. 23–30 (2011)
15. Gavrilut, D., Cimpoes, M., Anton, D., Ciortuz, L.: Malware Detection Using Machine Learning. In: Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 735–741 (2009) ISBN 978-83-60810-22-4
16. Ye, Y., Wang, D., Li, T., Ye, D.: Imds: intelligent malware detection system. ACM (2007)
17. Chandrasekaran, M., Vidyaraman, V., Upadhyaya, S.J.: Spycon: Emulating user activities to detect evasive spyware, IPCCC. IEEE Computer Society, 502–550 (2007)
18. Chouchane, M.R., Walenstein, A., Lakhotia, A.: Using Markov Chains to filter machine-morphed variants of malicious programs. In: 3rd International Conference on Malicious and Unwanted Software, MALWARE 2008, pp. 77–84 (2008)
19. Stamp, M., Attaluri, S.: McGhee S.: Profile hidden markov models and metamorphic virus detection. Journal in Computer Virology (2008)
20. Santamarta, R.: Generic detection and classification of polymorphic malware using neural pattern recognition (2006)
21. Yoo, I.: Visualizing Windows executable viruses using self-organizing maps. In: VizSEC/DMSEC 2004: ACM Workshop (2004)
22. Schliebs, S., Kasabov, N.: Evolving spiking neural network—a survey. Evolving Systems 4(2), 87–98 (2013)
23. Thorpe, S.J., Delorme, A.: Rufin van Rullen: Spike-based strategies for rapid processing. Neural Networks 14(6-7), 715–725 (2001)
24. Delorme, A., Perrinet, L., Thorpe, S.J.: Networks of Integrate-and-Fire Neurons using Rank Order Coding B: Spike Timing Dependant Plasticity and Emergence of Orientation Selectivity. Published in Neurocomputing 38-40(1-4), 539–545 (2000)
25. Thorpe, S.J., Gautrais, J.: Rank order coding. In: CNS 1997: Proceedings of the 6th Annual Conference on Computational Neuroscience: Trends in Research, New York, NY, USA, pp. 113–118. Plenum Press (1998)
26. Kasabov, N.: Evolving connectionist systems: Methods and Applications in Bioinformatics. In: Yu, P.X., Kacprzyk, P.J. (eds.) Brain Study and Intelligent Machines. Springer, NY (2002)

27. Wysoski, S.G., Benuskova, L., Kasabov, N.: Adaptive learning procedure for a network of spiking neurons and visual pattern recognition. In: Blanc-Talon, J., Philips, W., Popescu, D., Scheunders, P. (eds.) ACIVS 2006. LNCS, vol. 4179, pp. 1133–1142. Springer, Heidelberg (2006)
28. Schliebs, S., Defoin-Platel, M., Kasabov, N.: Integrated feature and parameter optimization for an evolving spiking neural network. In: Köppen, M., Kasabov, N., Coghill, G. (eds.) ICONIP 2008, Part I. LNCS, vol. 5506, pp. 1229–1236. Springer, Heidelberg (2009)
29. Song Q., Kasabov N.: Weighted Data Normalization and Feature Selection. In: Proc. of the 8th Intelligence Information Systems Conference (2003)
30. Huang, L., Song, Q., Kasabov, N.: Evolving Connectionist System Based Role Allocation for Robotic Soccer. International Journal of Advanced Robotic Systems 5(1), 59–62 (2008) ISSN 1729-8806
31. Kasabov, N.: Evolving fuzzy neural networks for online supervised/ unsupervised, knowledge–based learning. IEEE Trans. Cybernetics 31(6), 902–918 (2001)
32. Kasabov, N., Song, Q.: DENFIS: Dynamic, evolving neural-fuzzy inference systems and its application for time-series prediction. IEEE Trans. 10(2), 144–154 (2002)
33. Goh, L., Song, Q., Kasabov, N.: A Novel Feature Selection Method to Improve Classification of Gene Expression Data. In: 2nd Asia-Pacific IT Conf. vol. 29 (2004)
34. Kasabov, N., Song, Q.: GA-parameter optimization of evolving connectionist systems for classification and a case study from bioinformatics. In: Neural Information ICONIP 2002 Proceedings of the 9th International Conference on, IEEE ICONIP, 1198128 (2002)
35. http://www.kedri.aut.ac.nz/
36. http://malfease.oarci.net
37. http://roberto.perdisci.googlepages.com/code
38. http://www.cs.waikato.ac.nz/ml/weka