A Bio-Inspired Hybrid Artificial Intelligence Framework for Cyber Security

Konstantinos Demertzis and Lazaros Iliadis

Abstract Confidentiality, Integrity, and Availability of Military information is a crucial and critical factor for a country's national security. The security of military information systems (MIS) and Networks (MNET) is a subject of continuous research and design, due to the fact that they manage, store, manipulate, and distribute the information. This study presents a bio-inspired hybrid artificial intelligence framework for cyber security (bioHAIFCS). This framework combines timely and bio-inspired Machine Learning methods suitable for the protection of critical network applications, namely military information systems, applications and networks. More specifically, it combines (a) the hybrid evolving spiking anomaly detection model (HESADM), which is used in order to prevent in time and accurately, cyber-attacks, which cannot be avoided by using passive security measures, namely: Firewalls, (b) the evolving computational intelligence system for malware detection (ECISMD) that spots and isolates malwares located in packed executables untraceable by antivirus, and (c) the evolutionary prevention system from SQL injection (ePSSQLI) attacks, which early and smartly forecasts the attacks using SQL Injections methods.

1 Introduction

Application of high protection level measures by the army, in order to secure its information systems (IS), can offer a serious advantage in the evolution of a crisis, in tactical and operational level. It is a fact that the necessity to ensure secrecy of military IS and Confidentiality of information control and management systems (C4I) is a critical stabilization factor between opposite forces and a matter of honor for each side. The opposite can have serious consequences difficult to estimate in terms of material or moral cost. Thus, the development of network security systems following military specifications and demands is absolutely necessary. They could combine smart techniques capable of preventing attacks of zero-day nature.

K. Demertzis (✉) • L. Iliadis
Department of Forestry & Management of the Environment & Natural Resources, Democritus University of Thrace, Xanthi 671 00, Greece
e-mail: kdemertz@fmenr.duth.gr; liliadis@fmenr.duth.gr

N.J. Daras, M.Th. Rassias (eds.), *Computation, Cryptography, and Network Security*, DOI 10.1007/978-3-319-18275-9_7

The most popular attack techniques aiming to gain access in important or sensitive data use one of the methods below:

- Direct invasion to the system with attacks of DoS,
- Dispersion and installation of malware software
- Exploitation of potential weaknesses in the security of the system and mainly in the security of the network applications with attacks of SQL Injections type.

In the case of direct attack in a network, the usual security measures are the installation of a Firewall, in order to prevent non-authorized access in certain services and the installation of an intrusion detection system (IDS). The IDS are network and event monitoring and analysis systems. The target is to spot indications of potential intrusion efforts or efforts aiming to deviate the security mechanisms by external non-authorized users or users with limited authorization. The protection in this case is based on passive measures that use statistical analysis of events. There are network based (NIDS) and host based (HIDS) IDSs. Some of them are looking for specific signatures of known threats, whereas others are spotting anomalies by comparing traffic patterns against a baseline [1].

There are three basic approaches for designing and building IDS, namely: the Statistical, the Knowledge based, and the Machine Learning one which has been employed in this research effort. The concept of the statistical-based systems (SBID) is simple: it determines "normal" network activity and then all traffic that falls outside the scope of normal is flagged as anomalous (abnormal). These systems attempt to learn network traffic patterns on a particular network. This process of traffic analysis continues as long as the system is active, so, assuming network traffic patterns remain constant, the longer the system is on the network, the more accurate it becomes. The knowledge based intrusion detection systems (KBIDES) classify the data vectors based on a carefully designed Rule Set or they use models obtained from past experience in a heuristic mode. The Machine Learning approach automates the analysis of the data vectors, and they result in the implementation of systems that have the capacity to improve their performance as time passes.

This research effort aims in the development and application of an innovative hybrid evolving spiking anomaly detection model (HESADM) [2], which employs classification performed by evolving spiking neural networks (eSNN), in order to properly label a potential anomaly (PAN) in the net. On the other hand, it uses a multi-layer feed forward (MLFF) ANN to classify the exact type of the intrusion.

The second attack approach is the dispersion and installation of malwares which are untraceable by the usual antivirus systems. Malware is a kind of software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems. 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 [3].

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 [4].

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) [5] approach for the identification of packed executables and detection of malware by employing eSNN. A multilayer evolving classification function (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.

The third way widely used to overcome the security measures by exploiting the gaps in the control systems is the SQL injections one. This approach tries to exploit vulnerabilities in the security of network applications. SQL injection is a code injection technique, used to attack data driven applications, in which malicious SQL statements area SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system, and in some cases issue commands to the operating system. SQL injection

attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands [6]. This study proposes a bio-inspired Artificial Intelligence model named evolutionary prevention system from SQL injection (ePSSQLI) Attacks. It combines the use of MLFF ANN with optimization techniques of genetic algorithms (evolutionary optimization), in order to handle the potential intrusion attacks, based on SQL injection type.

2 Literature Review

Artificial Intelligence and data mining algorithms have been applied as intrusion detection methods in finding new intrusion patterns [7–10], such as clustering (unsupervised learning) [11–13] or classification (supervised learning) [14–17]. Also, a few hybrid techniques were proposed like Neural Networks with Genetic Algorithms [18] or Radial Based Function Neural Networks with Multilayer Perceptron [19, 20]. Besides, other very effective methods exist such as Sequential Detection [21], State Space [22], Spectral Methods [23], and combinations of those.

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 [24]. 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 [25] Renovo [26], OmniUnpack [27], or Eureka [28].

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 [29]. 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 [30], PE-Probe [31], and Perdisci et al. [32]). 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 [33–35].

Artificial Intelligence and data mining algorithms have been applied as malicious detection methods and for the discovery of new malware patterns [36]. In the research effort of Babar and Khalid [29], boosted decision trees working on n-grams are found to produce better results than Naive Bayes classifiers and support vector machines (SVMs). Ye et al. [37] use automatic extraction of association rules on Windows API execution sequences to distinguish between malware and clean program files. Chandrasekaran et al. [38] used association rules, on honeytokens

of known parameters. Chouchan et al. [39] 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. [40] employ profile hidden Markov Models, which have been previously used for sequence analysis in bioinformatics. Artificial Neural Networks (ANN) to detect polymorphic malware is explored in [41]. Yoo [42] employs Self-Organizing Maps to identify patterns of behavior for viruses in Windows executable files. These methods have low accuracy as a consequence, packed benign executables would likely cause false alarm, whereas malware may remain undetected.

Vulnerability pattern approach is used by Livshits et al. [43], they propose static analysis approach for finding the SQL injection attack. The main issue of this method is that it cannot detect the SQL injection attack patterns that are not known beforehand. Also, AMNESIA mechanism to prevent SQL injection at run time is proposed by Halfond et al. [44]. It uses a model based approach to detect illegal queries before it sends for execution to database. The mechanism which filters the SQL Injection in a static manner is proposed by Buehrer et al. [45]. The SQL statements by comparing the parse tree of a SQL statement before and after input and only allowing to SQL statements to execute if the parse trees match. Marco Cova et al. [46] proposed a Swaddler, which analyzes the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state.

There exists machine learning related works in the wild [47–51]. In this work we focus on the detection at the spot between application and database, detecting anomalous SQL statements (the SQL statement returns a result set of records from one or more tables), which are malicious in the sense that they include parts of injected code or differ from the set of queries usually issued within an application. Valeur et al. [52] proposed the use of an IDS based on a machine learning technique which identifies queries that do not match multiple models of typical queries at runtime, including string model and data type-independent model. It is trained by a set of typical application queries, and the quality depends on the quality of the training set. Wang et al. presented a novel method for learning SQL statements and apply machine learning techniques, such as one class classification, in order to detect malicious behavior between the database and application [53]. The approach incorporates the tree structure of SQL queries as well as input parameter and query value similarity as characteristic to distinguish malicious from benign queries. Rawat et al. use SVM for classification and prediction of SQL-Injection attack [54]. This work contains the idea that compares SQL query strings and blocks suspicious SQL-query and passes original SQL-query. Huang et al. present a new method to prevent SQLI attack based on machine learning [55]. This approach identifies SQL injection codes by HTTP parameters' attributes and the Bayesian classifier. This technique depends on the choices of patterns' attributes and the quality of the training set. They choose two values as attributes of patterns, and invent a way to generate the real-world patterns automatically. In addition Huang et al. designed a system based on machine learning for preventing SQL injection attack, which utilizes pattern classifiers to detect injection attacks and protect web applications [56]. The system captures parameters of HTTP requests, and converts them into

numeric attributes. Numeric attributes include the length of parameters and the number of keywords of parameters. Using these attributes, the system classifies the parameters by Bayesian classifier for judging whether parameters are injection patterns.

3 Methodologies Comprising the bioHAIFCS

The bioHAIFCS uses three biologically inspired Artificial Intelligence methods, namely: eSNN, MLFF, and ECF and their corresponding optimization approach with GA, in order to create a high level security framework. It acts in a smart and preemptive manner to spot the threats by making the minimum consumption of resources. These methods are presented below:

3.1 Evolving Spiking Neural Networks

The eSNN that has been developed and discussed herein is based on the *"Thorpe"* neural model [57] which 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 [58, 59]. 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 details of eSNN architecture described below:

3.1.1 Rank Order Population Encoding

The ROPE method [58, 59] is an alternative to conventional rate coding scheme that uses the order of firing neuron's inputs to encode information which allows the mapping of vectors of real-valued elements into a sequence of spikes. Neurons 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, neurons are progressively desensitized each time one of their inputs fires, thus making neuronal responses dependent upon the relative order of firing of the neuron's afferents. More precisely, let $A = \{a_1, a_2, a_3 \ldots a_{m-1}, a_m\}$ be

the ensemble of afferent neurons of neuron i and $W = \{w_{1,i}, w_{2,i}, w_{3,i} \ldots w_{m-1,i}, w_{m,i}\}$ the weights of the m corresponding connections; let mod $\in [0,1]$ be an arbitrary modulation factor. The activation level of neuron i at time t is given by Eq. (1):

$$\text{Activation}(i,t) = \sum_{j \in [1,m]} \text{mod}^{\text{order}(aj)} w_{j,i} \tag{1}$$

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. Whenever 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 particular location. Every spike also 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.

$$\text{if } t_e < t_a \text{ then } dW = a(1\text{-}W)e^{-|\Delta o|\tau} \tag{2}$$

$$\text{else } dW = -aWe^{-|\Delta o|\tau}$$

where Δo is the difference between the date of the EPSP and the date of the neuronal discharge (expressed in terms of order of arrival instead of time), as is a constant that controls the amount of synaptic potentiation and depression [58].

ROPE technique with receptive fields allows the encoding of continuous values by using a collection of neurons with overlapping sensitivity profiles [60]. Each input variable is encoded independently by a group of one-dimensional receptive fields (Fig. 2). 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:

$$\mu i = I^n_{min} + \frac{2i\text{-}3}{2} \frac{I^n_{max} - I^n_{min}}{M\text{-}2} \tag{3}$$

The width σ is given by Eq. (4):

$$\sigma = \frac{1}{\beta} \frac{I^n_{max} - I^n_{min}}{M\text{-}2} \tag{4}$$

where $1 \leq \beta \leq 2$ and the parameter β directly controls the width of each Gaussian receptive field.

Figure 1 depicts an example encoding of a single variable. For the diagram ($\beta = 2$) the input interval $[I^n_{min}, I^n_{max}]$ was set to $[-1.5, 1.5]$ and M=5 receptive fields were used. For an input value v=0.75 (thick straight line in left figure) the intersection points with each Gaussian is computed (triangles), which are in turn translated into spike time delays (right figure).

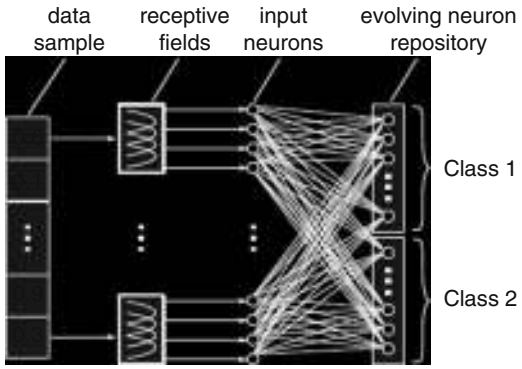

Fig. 1 The evolving spiking neural network (eSNN) architecture [23]

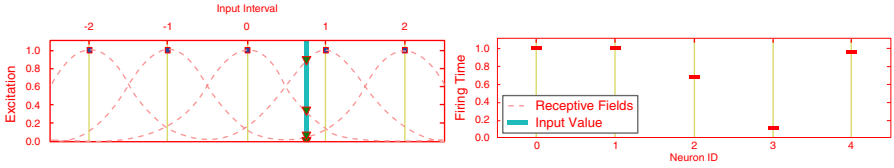

Fig. 2 Population encoding based on Gaussian receptive fields [23]

3.1.2 One-Pass Learning

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 SANN which may result in the firing of certain output neurons. It is also possible that no output neuron is activated and in this case 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 represents the classification result for the presented input sample. The procedure is described in detail in the following Algorithm 1 [23, 60] (Fig. 2).

For each training sample i with class label $l \in L$ 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: $w_j^{(i)}=(m_l)^{\text{order(j)}}$, $\forall j|j$ 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 $u_{max}^{(i)}$:

Algorithm 1 Training an Evolving Spiking Neural Network (eSNN) [23]

Require: m_l, s_l, c_l **for a class label** l $\in$ L
initialize neuron repository $R_l = \{\}$
for all samples $X^{(i)}$ belonging to class l **do**
  $w_j^{(i)} \leftarrow (m_l)^{order(j)}$, $\forall$ j | j pre-synaptic neuron of i
  $u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)}$
  $\theta^{(i)} \leftarrow c_l u_{max}^{(i)}$
if min(d($w^{(i)}$, $w^{(k)}$)) < s_l, $w^{(k)} \in R_l$ **then**
  $w^{(k)} \leftarrow$ merge $w^{(i)}$ and $w^{(k)}$ according to Eq. (6)
  $\theta^{(k)} \leftarrow$ merge $\theta^{(i)}$ and $\theta^{(k)}$ according to Eq. (7)
else
  $R_l \leftarrow R_l \cup \backslash\backslash \{w^{(i)}\backslash\backslash\}$
end if
end for

$$\theta^{(i)} \leftarrow c_l u_{max}^{(i)} \tag{5}$$

$$u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)} \tag{6}$$

The fraction c_l is a parameter of the model and for each class label l $\in$ L a different fraction can be specified. The weight vector of the trained neuron is then 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 modulation factor m_l, similarity threshold s_l, PSP fraction c_l, l $\in$ L of ESNN which were included in this search space, are optimized according to the versatile quantum-inspired evolutionary algorithm (vQEA) [61]. In this case, both the firing thresholds and the weight vectors are merged according to Eqs. (7) and (8):

$$w_j^{(k)} \leftarrow \frac{w_j^{(i)} + N w_j^{(k)}}{1+N}, \forall j \mid j \text{ pre-synaptic neuron of i} \tag{7}$$

$$\theta^{(k)} \leftarrow \frac{\theta^{(i)} + N\theta^{(k)}}{1+N} \tag{8}$$

It must be clarified that 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. After the 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 neuron.

3.2 Multilayer Feed-Forward Neural Network

Artificial neural networks are biologically inspired classification algorithms that consist of an input layer of nodes, one or more hidden layers, and an output layer. Each node in a layer has one corresponding node in the next layer, thus creating the stacking effect [62]. Artificial neural networks are the very versatile tools and have been widely used to tackle many issues [63–67].

Feed-forward neural networks (FNN) are one of the popular structures among artificial neural networks. These efficient networks are widely used to solve complex problems by modeling complex input–output relationships [68, 69]. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.

The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoidal functions.

Feed-forward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors. The linear output layer is most often used for function fitting (or nonlinear regression) problems.

Multi-layer networks use a variety of learning techniques, the most popular being back-propagation. Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has learned a certain target function. To adjust weights properly, one applies a general method for nonlinear optimization that is called gradient descent. For this, the derivative of the error function with respect to the network weights is calculated, and the weights are then changed such that the error decreases (thus going downhill on the surface of the error function).

3.3 Evolving Connectionist Systems

Evolving connectionist systems (ECOS) [70] are multi-modular, connectionist architectures that facilitate modeling of evolving processes and knowledge discovery [60]. An ECOS may consist of many evolving connectionist modules. An ECOS

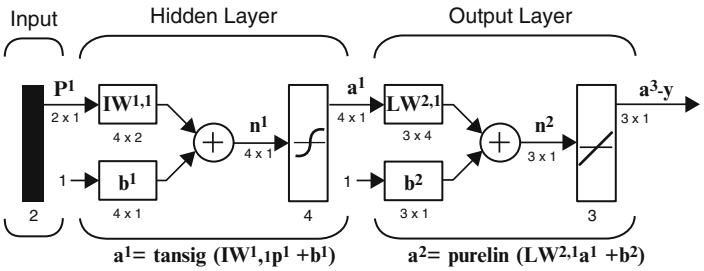

Fig. 3 Architecture of the multilayer feed-forward artificial neural network (http://www.mathworks.com/)

is a neural network that operates continuously in time and adapts its structure and functionality through a continuous interaction with the environment and with other systems according to:

- a set of parameters that are subject to change during the system operation;
- an incoming continuous flow of information with unknown distribution;
- a goal (rational) criterion (also subject to modification) that is applied to optimize the performance of the system over time.

The ECOS evolve in an open space, using constructive processes, not necessarily of fixed dimensions. Moreover, 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. They learn locally and locally partition the problem space, thus allowing for a fast adaptation and tracing processes over time. They facilitate different kinds of knowledge representation and extraction, mostly—memory based statistical and symbolic knowledge [60, 71, 72] (Fig. 3).

ECOS are connectionist structures that evolve their nodes (neurons) and connections through supervised incremental learning from input–output data pairs.

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 [60, 71, 72].

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 W1 connection weights of these nodes represent the coordinates of the nodes in the input space. The second layer W2 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 Rmax are allocated to the same cluster Nc. 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 = (W1, W2) can be measured in different ways, the most popular of them being the normalized Euclidean distance:

$$d(S, N) = \frac{1}{n}\left[\sum_{i=1}^{n} |x_i\text{-}W_{1N}|^2\right]^{\frac{1}{2}} \tag{9}$$

where n is the number of the input variables.

ECOS learn from data and automatically create a local output function for each cluster, the function being represented in the W2 connection weights, thus 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.

The following is a corresponding example of such a local Rule:

- IF (data is in cluster Nc), THEN (the output is calculated with a function Fc)
- In the case of DENFIS [32], first order local fuzzy rule models are derived incrementally from data. The following rule is a characteristic example:
- IF (the value of x1 is in the area defined by a Gaussian function with a center at 0.7 and a standard deviation of 0.1) AND (the value of x2 is in the area defined by a Gaussian function with a center at 0.5 and a deviation of 0.2), THEN (the output value y is calculated with the use of the formula y= 3.7 + 0.5x1−4.2x2).

3.3.1 Evolving Classification Function

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 center, 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 fourth layer represents classes [60, 70–72].

3.4 Genetic Algorithm

The genetic algorithm (GA) is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution (http://www.mathworks.com/). The GA repeatedly modifies a population of individual solutions. At each step, the GA selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the GA to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. Also the GA can address problems of mixed integer programming, where some components are restricted to be integer-valued.

The GA uses three main types of rules at each step to create the next generation from the current population:

- Selection rules select the individuals, called parents, that contribute to the population at the next generation.
- Crossover rules combine two parents to form children for the next generation.
- Mutation rules apply random changes to individual parents to form children.

The GA differs from a classical, derivative-based, optimization algorithm in two main ways, as follows:

- Classical Algorithm
  - Generates a single point at each iteration. The sequence of points approaches an optimal solution.
  - Selects the next point in the sequence by a deterministic computation.
- Genetic Algorithm
  - Generates a population of points at each iteration. The best point in the population approaches an optimal solution.
  - Selects the next population by computation which uses random number generators.

3.4.1 Genetic Algorithm for Offline ECF Optimization

A GA is applied to a population of solutions to a problem 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 (loosely analogous to those operations found in biological systems). After several generations, the population of solutions should converge on a "good" solution.

Given that the ECF system is a neural network that operates continuously in time and adapts its structure and functionality through a continuous interaction with the environment and with other systems 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 (also subject to modification) that is applied to optimize the performance of the system over time.

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 whole 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 [73]. 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 [60, 70–72].

4 Description of the Proposed Hybrid Framework

Considering that the aim of the partial proposed systems is to carry out acts in a common environment, the architecture of the bioHAIFC can be simulated by a distributed multi-agent AI system. The agents are the three proposed Machine Learning systems, namely: (HESADM, ECISMD and ePSSQLI). These systems dynamically control the predefined sectors with a potential threat [74]. The synchronization of the Agents is achieved either with negotiation or with cooperation, as none of them has the full information package, there is no central control in the system, the data are distributed and the calculations are done in an asynchronous manner. The Agent communication and information exchange is done by a hybrid system of temporal programming in order to phase (in an optimal way) the potential contradiction of intensions and contradiction in the management of resources, based on priorities related to the extent of the threat and risk.

The results of the characterization of a threat are sent to the administrator of the network in a form of logs. The administrator tries to take necessary prevention actions in order to avoid the risk. Also the framework automates the potential direct termination of the TCP connection operation with the attacker for higher security and control (e.g., tcpkill host 192.168.1.2 or tcpkill host host12.blackhut.com).

The analytical description of the partial systems of the bioHAIFCS is described below:

4.1 Hybrid Evolving Spiking Anomaly Detection Model

The HESADM methodology uses eSNN classification approach and Multi-Layer Feed Forward ANN in order to classify the exact type of the intrusion or anomaly in the network with minimum computational power. The dataset which used and the general algorithm are described in detail below:

4.1.1 Data

The *KDD Cup* 1999 data set [75] was used to test the herein proposed approach. This data set was created in the LincolnLab of MIT and it is the most popular free data set used in evaluation of IDS. It contains recordings of the total network flow of a local network which was installed in the Lincoln Labs and it simulates the military network of the USA air force. The method of events' analysis includes a connection between a source IP address and a destination IP, during which a sequence of TCP packages is exchanged, by using a specific protocol and a strictly defined operation time.

The KDD Cup 1999 data includes 41 characteristics which are organized in the following four basic categories: Content Features, Traffic Features, Time-based Traffic Features, Host-based Traffic Features. Also the attacks are divided into four categories, namely: DoS, r2l, u2r, and probe.

Using the eSNN Traf_Red_Full.data In the first classification case, all (41) features were used. The data were classified as normal or abnormal. The dataset Traf_Red_Full.data has 145,738 records and the 75 % (109,303 rec.) used as train_data and the 25 % (36,435 rec.) used as test_data.

Using the SNN normalFull.data In the second classification case, the relevant normal features comprising of 11 features were used. The data were classified as normal or abnormal. The dataset normalFull.data has 145,738 records and the 75 % (109,303 rec.) were used as train_data and the 25 % (36,435 rec.) as test_data.

4.1.2 Algorithm

- **Step 1**

We choose to use the traffic oriented data, which is related to only nine features. We import the required classes that use the variable Population Encoding. 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). We also normalize the data to the interval [−1,1] and so we indicate the coverage of the Gaussians using i_min and i_max. For the normalization processing the following function 10 was used:

$$x_{1_{\text{norm}}} = 2* \left(\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}} \right) - 1, x \in R \tag{10}$$

The data is classified into two classes namely: class 0 which contains the normal results and class 1 which comprises of the abnormal ones (DoS, r2l, u2r and probe). The eSNN object using modulation factor m=0.9, firing threshold ratio c=0.7 and similarity threshold s=0.6 in agreement with the vQEA algorithm [23, 61].

- **Step 2**

We train the eSNN with 75 % of the dataset vectors (*train_data) a*nd we test the eSNN with 25 % of the dataset vectors *(test_data).* The training process is described in Algorithm 1.

- **Step 3**

If the result of the classification is normal, the eSNN classification process is repeated but this time the relevant normal data vectors are used. These vectors are comprised of 11 features [9]. If the result is normal, then the process is terminated. If the result of the classification is abnormal, a two-layer feed-forward neural network with sigmoid function both in hidden and output layer with scaled conjugate gradient backpropagation as the learning algorithm is used to perform pattern recognition of the attack type with all features of KDD dataset (41 inputs and 5 outputs).

The outcome of the pattern recognition process is submitted in the form of an *Alert* signal to the network administrator. A Graphical display of the complete HESADM methodology can be seen in Fig. 4.

The performance metric used is the mean squared error (MSE). The MLFF ANN was developed with 41 input neurons, corresponding to the 41 input parameters of the KDD cup 1999 dataset, 33 neurons in the Hidden Layer, and 5 in the output one corresponding to the following output parameters: DoS, r2l, u2r, Probe, normal. In the hidden layer 33 neurons are used, based on the following empirical function 11 [76]:

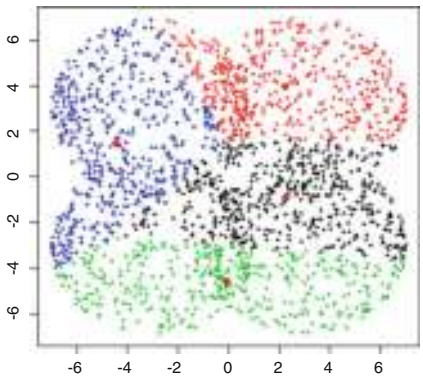

Fig. 4 Rule of the evolving connectionist system [60, 70–72]

$$\left(\frac{2}{3}*\text{Inputs}\right) + \text{Outputs} = \left(\frac{2}{3}*41\right) + 5 = 33 \tag{11}$$

The KDD cup 1999 dataset was divided randomly into 70 % (102,016 rec.) the train_data, 15 % (21,861 rec.) as test_data and the rest 15 % (21,861 records) as validation_data.

4.2 *Evolving Computational Intelligence System for Malware Detection*

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 (Fig. 5).

The dataset which used and the general algorithm are described below:

4.2.1 Dataset

The full_dataset comprised of 2598 packed viruses from the Malfease Project dataset (http://malfease.oarci.net), 2231 non-packed benign executables collected from a clean installation of *Windows XP Home plus*, several common user applications and 669 packed benign executables.

The dataset was divided randomly into two parts:

- A training dataset containing 2231 patterns related to the non-packed benign executable and 2262 patterns related to the packed executables detected using unpacked software
- A testing dataset containing 1005 patterns related to the packed executables that even the best known unpacked software was not able to detect. These datasets are available at http://roberto.perdisci.googlepages.com/code [32].

The virus dataset containing 2598 malware and 669 benign executables is divided into two parts:

- A training dataset containing 1834 patterns related to the malware and 453 patterns related to the benign executables
- 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. [32] 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.

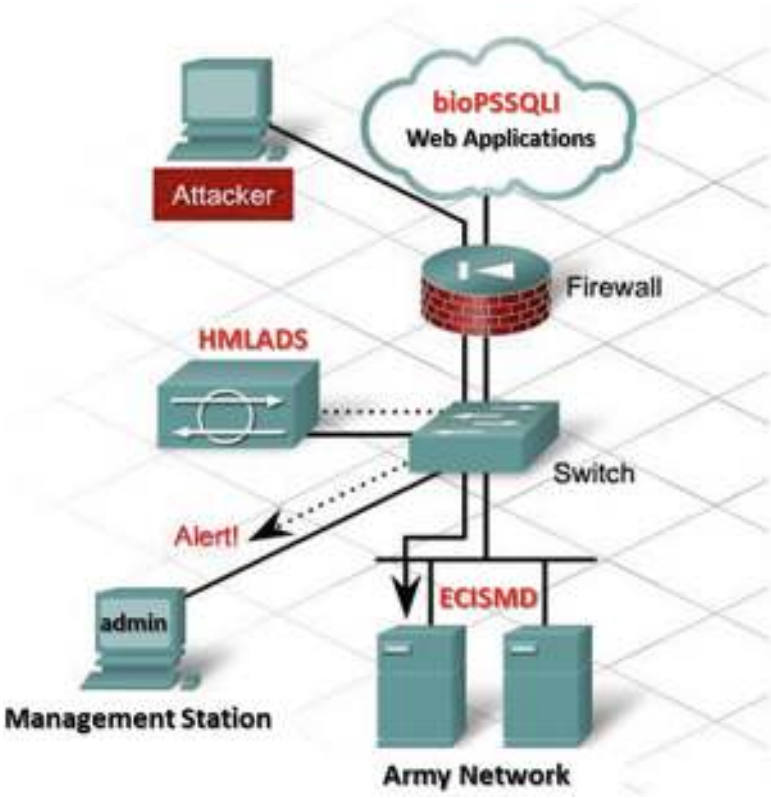

Fig. 5 Bio-inspired hybrid artificial intelligence framework for cyber security

In the first classification performed by the ECISMD, the eSNN approach was employed in order to classify packed or not packed executables.

In the second classification performed by the ECISMD, the ECF approach was employed in order to classify malware or benign executables.

4.2.2 Algorithm

- **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 function 10 is used (Fig. 6).

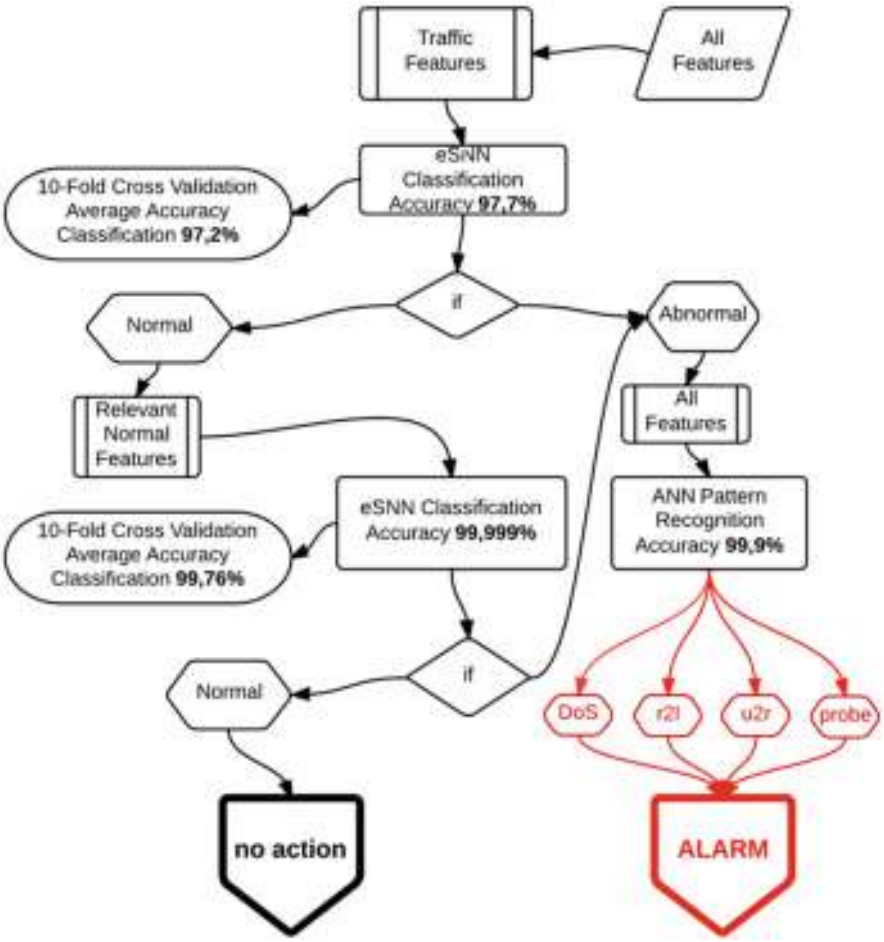

Fig. 6 The hybrid evolving spiking anomaly detection model (HESADM) methodology

The data is classified into two classes, namely: ***Class 0*** which contains the unpacked results and ***Class 1*** which comprises of the packed ones. The eSNN object using modulation factor m=0.9, firing threshold ratio c=0.7, and similarity threshold s=0.6 in agreement with the vQEA algorithm [23, 61].

- **Step 2**

The eSNN is trained with the *packed_train* dataset vectors and the testing is performed with the *packed_test* vectors. The training process is described in Algorithm 1.

- **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 nine features and two classes malware and benign.

The learning algorithm of the ECF according to the ECOS is as follows:

- 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.
- 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.
- 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.
- 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.
- 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 [77].

- **Step 4**

To increase the level of integrity the Offline ECF Optimization with GA is used.

- **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 unpacker, to the antivirus scanner for verification and finally placed in quarantine and the process is terminated (Fig. 7).

4.3 Evolutionary Prevention System from SQL Injection

The proposed ePSSQLI model uses an MFFNN which has optimized with a GA. Generally, there are three methods of using a GA for training MFFNNs. Firstly, GA is utilized for finding a combination of weights and biases that provide the minimum

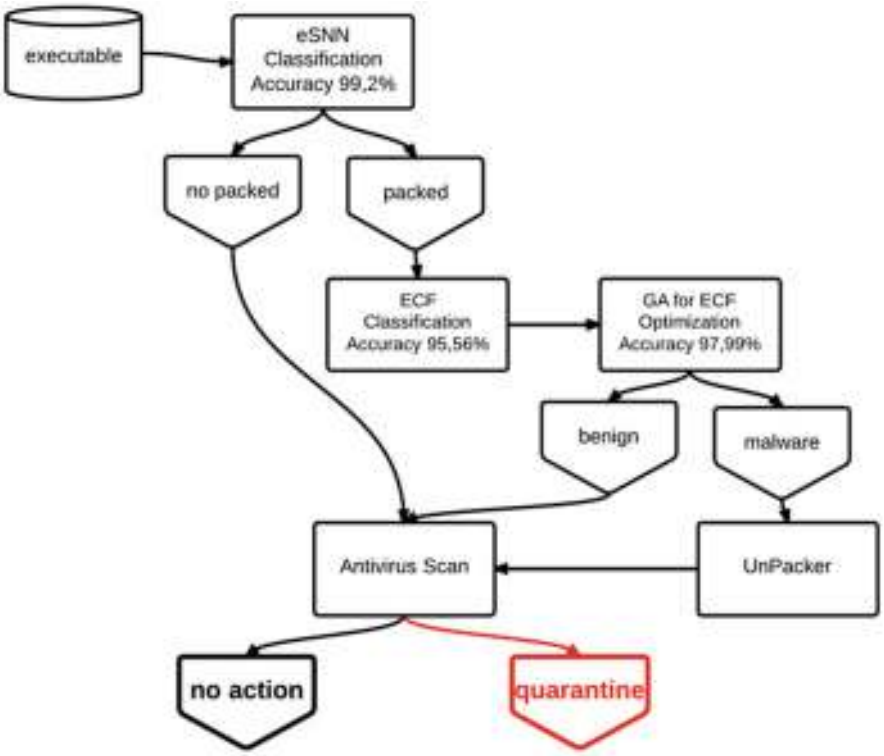

Fig. 7 Graphical display of the ECISMD algorithm

error for an MFFNN. Secondly, GA is employed to find a proper architecture for an MFFNN in a particular problem. The last method is to use a GA to tune the parameters of a gradient-based learning algorithm, such as the learning rate and momentum. In the first method, the architecture does not change during the learning process. The training algorithm is required to find proper values for all connection weights and biases in order to minimize the overall error of the MFFNN. In the second approach, the structure of the MFFNNs varies. In this case, a training algorithm determines the best structure for solving a certain problem. Changing the structure can be accomplished by manipulating the connections between neurons, the number of hidden layers, and the number of hidden nodes in each layer. In this study the GA is applied to minimize the error of MFFNN in order to classify SQL injections with high accuracy.

The dataset which used and the general algorithm are described below:

4.3.1 Dataset

The dataset used includes a list of 13,884 SQL statements that have been selected by various sources. Actually, 12,881 of them are malicious (SQL Injections) and 1003 are legit. With the help of the SQLparse module (https://github.com/andialbrecht/sqlparse) in Python, which is a non-validating SQL one, we have searched the way of syntax and use of certain SQL symbols in the construction of SQL injections commands. Also we investigated the correlation of SQL statements with the attacks of SQL injections' type.

Finally, the n-gram technique was used to search the correlation of the SQL statements sequence, with the syntax of the SQL injections commands (https://github.com/ClickSecurity/data_hacking). In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words, or base pairs according to the application. The n-grams in this case are collected from an SQL statements.

Various malicious και legit scores constitute the statistical output of the SQL statements and they were used as features. In information theory, entropy is a measure of the uncertainty associated with a random variable. The term by itself in this context usually refers to the Shannon entropy, which quantifies, in the sense of an expected value, the information contained in a message, usually in units such as bits. Equivalently, the Shannon entropy is a measure of the average information content one is missing when one does not know the value of the random variable [78].

After its adjustment, the dataset includes the following parameters:

- Length
- Entropy
- Malicious_score
- Legit_score
- Difference_score
- Class

In the pre-processing of data remove extreme values and outliers. The extreme value is a point which is far away from the average value of a parameter. The distance is measured based on a threshold which is a multiplicand of the standard deviation (Fig. 8).

We know that for a random parameter that is under normal distribution, the 95 % of all the values fall up to the value of 2*stdev whereas 99 % fall up to the value of 3*stdev. Extreme values cause significant errors in a potential model. Things become even worse when these extreme values are noise results during measurements procedure. If the number of extreme values is small, then they are removed from the data set.

The estimation of the extreme values was done under the Inter Quartile Range method [79]. This method spots extreme values and outliers based on (InterQuartile Ranges—IQR). The IQR is the difference between the third (Q3) and the first (Q1)

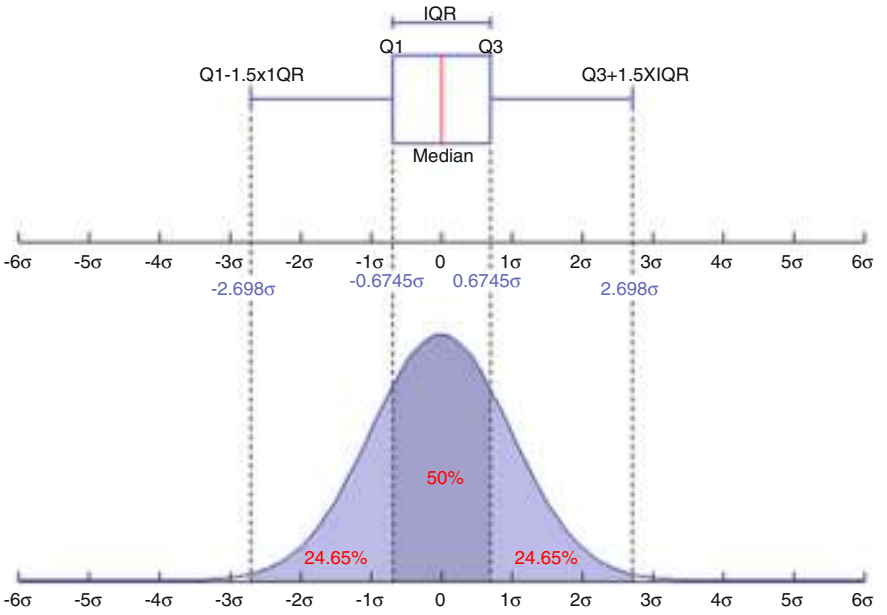

Fig. 8 Graphical display of inter quartile range method

quartile, IQR= Q3 – Q1. The quartiles divide the data into four equal parts. The IQR includes the imtermediate 50 % of the data whereas the rest 25 % is less than Q1 and the rest 25 % is higher than Q3 [2]. The calculation of the Extreme values was done as follows:

- Outliers:
  - Q3 + OF*IQR < x <= Q3 + EVF*IQR or Q1 - EVF*IQR <= x < Q1 - OF*IQR
- Extreme values:
  - x > Q3 + EVF*IQR or x < Q1 - EVF*IQR

Key: Q1=25 % quartile, Q3=75 % quartile, IQR=Interquartile Range difference between Q1 and Q3, OF=Outlier Factor, EVF=Extreme Value Factor.

With the use of the above method 12 outliers and three extreme values were removed from the data set which was reduced to 13,869 cases (12,881 malicious, 988 legit).

Also the data were Normalized so that they can have the proper input for the Learning Algorithms in the interval [−1, +1].

After a relative observation we can realize that we have created an imbalanced dataset which includes 13,869 cases from which 12,881 are malicious and 988 legit (0.0723 %). Imbalanced data sets are a special case for classification problem where the class distribution is not uniform among the classes. Typically, they are composed

by two classes: The majority (negative) class and the minority (positive) class. The problem with class imbalances is that standard learners are often biased towards the majority class. That is because these classifiers attempt to reduce global quantities such as the error rate, not taking the data distribution into consideration. As a result examples from the overwhelming class are well classified whereas examples from the minority class tend to be misclassified.

To resolve the certain problem we use the technique synthetic minority over-sampling technique (SMOTE) in order to resample the dataset. [80]. Re-sampling provides a simple way of biasing the generalization process. It can do so by generating synthetic samples accordingly biased and controlling the amount and placement of the new samples. SMOTE is a technique which combines Informed oversampling of the minority class with random undersampling of the majority class. SMOTE is a technique which is combines Informed oversampling of the minority class with random undersampling of the majority class and produce the best results as far as re-sampli ng and modifying the probabilistic estimate techniques.

For each minority sample, SMOTE works as follows:

- Find its k-nearest minority neighbors.
- Randomly select *j* of these neighbors.
- Randomly generate synthetic samples along the lines joining the minority sample and its *j* selected neighbors (*j* depends on the amount of oversampling desired).

By applying the SMOTE approach we re-created the dataset, which includes 21,773 cases, from which 12,881 are malicious and 8892 are legit.

4.4 Algorithm

The MLFF ANN was developed with five input neurons, corresponding to the five input parameters of the dataset, five neurons in the Hidden Layer and two in the output one corresponding to the following output parameters: malicious or legit. In the hidden layer five neurons are used, based on the empirical function 11.

This adds a greater degree of integrity to the rest of security infrastructure MFF ANN, optimized with GA. The following outline summarizes how the GA works:

- The algorithm begins by creating a random initial population.
- The algorithm then creates a sequence of new populations. At each step, the algorithm uses the individuals in the current generation to create the next population.
- To create the new population, the algorithm performs the following steps:
  - Scores each member of the current population by computing its fitness value.

- Scales the raw fitness scores to convert them into a more usable range of values.
- Selects members, called parents, based on their fitness.
- Some of the individuals in the current population that have lower fitness are chosen as *elite*. These elite individuals are passed to the next population.
- Produces children from the parents. Children are produced either by making random changes to a single parent—*mutation*—or by combining the vector entries of a pair of parents—*crossover*.
- Replaces the current population with the children to form the next generation.
- The algorithm stops when one of the stopping criteria is met.

5 Results

Each subsystem was tested based on multiple scenarios and different datasets were used for each case of threat. The results obtained are very encouraging as the accuracy is as high as 99 %, resulting in a reduction of the false alarms to the minimum. This fact, combined with the flexibility of the proposed system and with its generalization ability and the spotting of zero-day threats, makes its use suitable for critical applications like the one of military networks protection. The results of each case are presented below:

5.1 Hybrid Evolving Spiking Anomaly Detection Model

5.1.1 eSNN Approach

- In the first classification using the eSNN Traf_Red_Full.data the data classified as normal or abnormal. The results are shown below:
  - Classification Accuracy: 97.7 %
  - No. of evolved neurons: Class 0: 794 neurons, Class 1: 809 neurons
  - The average accuracy after applying tenfold Classification in the Traf_Red_Full.data was as high as 97.2 %.
- In the second classification case using the SNN normalFull.data, the relevant normal features comprising of 11 features were used. The data were classified as normal or abnormal. The results are shown below:
  - Classification Accuracy: 99.99 %
  - No. of evolved neurons: Class 0: 646 neurons, Class 1: 136 neurons
  - The average accuracy after applying tenfold Classification in the normalFull.data was as high as 99.76 %.

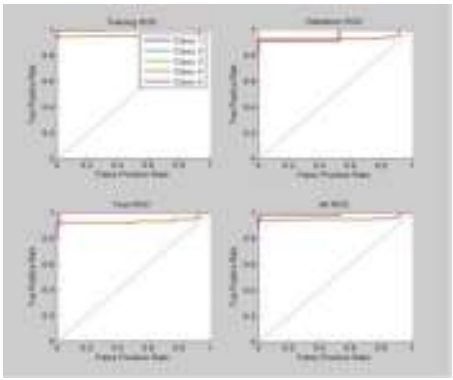

Fig. 9 ROC analysis

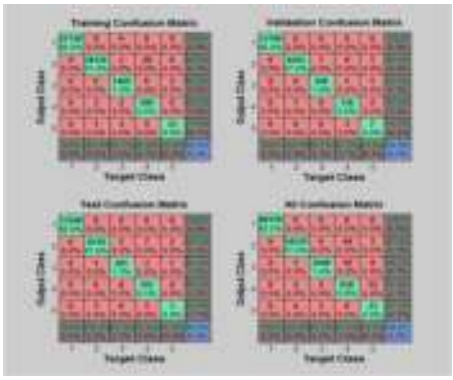

Fig. 10 Confusion matrix

5.1.2 MLFF ANN Approach

The classification accuracy is as high as 99.9% and all the performance metrics support the high level of convergence of the model.

In Fig. 9 the colored lines in each axis represent the ROC curves. The ROC curve is a plot of the true positive rate (sensitivity) versus the false positive rate (1-specificity) as the threshold is varied. A perfect test would show points in the upper-left corner, with 100 % sensitivity and 100 % specificity. For this problem, the network performs very well.

Figure 10 shows the confusion matrices for training, testing, and validation, and the three kinds of data combined. The network outputs are very accurate, by the high numbers of correct responses in the green squares and the low numbers of incorrect responses in the red squares. The lower right blue squares illustrate the overall accuracies.

5.2 ECISMD Results

Table 1 reports the average accuracy which computed over tenfold cross-validation obtained with RBF ANN, Naïve Bayes, multi layer perceptron (MLP), Support Vector Machine (SVM), k-Nearest-Neighbors (k-NN), and eSNN. The best results on the testing dataset were obtained by using the eSNN classifier, to classify packed or not packed executables.

Table 2 reports the results obtained with six 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 optimized ECF which classifies virus or benign executables (Table 3).

5.3 ePSSQLI Results

5.3.1 MFF ANN

The classification accuracy of the MFF ANN that uses tenfold Cross Validation before the optimization is equal to 97.7 %. The rest of the measurements and the confusion matrix are presented below (Table 4):

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

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

Table 3 Metrics of the MFF ANN

TP rate	FP rate	Precision	Recall	F-measure	ROC area	Class
0.986	0.034	0.976	0.986	0.981	0.986	Malicious
0.966	0.014	0.980	0.966	0.973	0.986	Legit

Table 4 Confusion matrix of the MFF ANN

Malicious	Legit
12,702	179
306	8586

Table 5 Metrics of the MFF ANN with GA

TP rate	FP rate	Precision	Recall	F-measure	ROC area	Class
0.997	0.003	0.998	0.997	0.997	0.998	Malicious
0.997	0.003	0.996	0.997	0.996	0.998	Legit

Table 6 Confusion matrix of the MFF ANN with GA

Malicious	Legit
12,845	36
31	8861

5.3.2 MFF ANN Optimized with GA

The initial parameters of GA are as below (Table 5):

- Selection: *Roulette wheel*
- Crossover: *Single point (probability = 1)*
- Mutation: *Uniform (probability = 0.01)*
- Population size: *200*
- Maximum number of generations: *250*

The classification accuracy of the MFF ANN that uses tenfold Cross Validation after its optimization with GA is 99.6 %. The rest of the measurements and the confusion matrix are presented below (Table 6):

The good performance and reliability of the proposed scheme that uses MFF ANN with GA is shown in Table 7 below. Table 7 presents the results of the categorization with the same dataset and by employing tenfold Cross Validation and other Machine Learning approaches.

6 Discussion: Conclusions

This paper proposes the use of a Bio-Inspired Hybrid Artificial Intelligence Framework for Cyber Security, which is based on the combination of three timely methods of Artificial Intelligence.

Table 7 Comparison of various approaches for the SQLI dataset

SQLI dataset	
Classifier	Accuracy (%)
MFF ANN with GA	99.6
RBFNetwork	97.3
fNaiveBayes	95.6
BayesNet	98.7
SVM	98.5
k-NN	98.3
Random forest	99.1

The function of the subsystems aims in the time spotting of the cyber-attacks which are untraceable with the classical passive protection approaches.

More specifically, this paper proposes the HESADM system, which spots potential anomalies of a network and the attacks that might bypass the firewall and the IDS. The second subsystem is ECISMD which scans the packed executable files and then spots malicious code untraceable by antivirus. The third one is ePSSQLI which spots in time the SQL Injections attacks. The result of each categorization is sent to the administrator of the system so that he/she can impose proper actions. An automatic disconnection from the attacker is also included.

The combination of the subsystems under the proposed framework takes place based on a temporal scheduling which succeeds the optimal distribution of the resources and the maximum availability and performance of the system. The use of the proposed systems can be done regardless of the framework.

The testing has resulted in an accuracy level of 99 %. Also a comparative analysis has revealed that the proposed algorithm outperforms the existing ones.

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. Also, the ECISMD could be improved towards a better online learning with self-modified parameter values. Finally, the MFF ANN with GA which used in the ePSSQLI system could be compared with other optimization schemes like particle swarm optimization.

References

1. Garcıa Teodoro, P., Dıaz-Verdejo, J., Macia-Fernandez, G., Vazquez, E.: Anomaly-based network intrusion detection: techniques, systems and challenges. Elsevier Comput. Security **28**, 18–28 (2009)
2. Demertzis, K., Iliadis, L.: A hybrid network anomaly and intrusion detection approach based on evolving spiking neural network classification. In: E-Democracy, Security, Privacy and Trust in a Digital World. Communications in Computer and Information Science, vol. 441, pp. 11–23. (2014). doi:10.1007/978-3-319-11710-2_2

3. Yan, W., Zhang, Z., Ansari, N.: Revealing packed malware. IEEE Secur. Priv. **6**(5), 65–69 (2007)
4. Cesare, S., Xiang, Y.: Software Similarity and Classification. Springer, New York (2012)
5. Demertzis, K., Iliadis, L.: Evolving computational intelligence system for malware detection. In: Advanced Information Systems Engineering Workshops. Lecture Notes in Business Information Processing, vol. 178, pp. 322–334. (2014). doi:10.1007/978-3-319-07869-4_30
6. Open Web Application Security Project (OWASP): (2014) https://www.owasp.org
7. Dorothy, D.E.: An intrusion-detection model. IEEE Trans. Softw. Eng. **13**, 222–232 (1987). doi:10.1109/TSE.1987.232894
8. Puketza, N., Zhang, K., Chung, M., Mukherjee, B., Olsson, R.A.: A methodology for testing intrusion detection system. IEEE Trans. Softw. Eng. **22**, 719–729 (1996). doi:10.1109/32.544350
9. *B*harti, K., Jain, S., Shukla, S.: Fuzzy K-mean clustering via random forest for intrusiion detection system. Int. J. Comput. Sci. Eng. **02**(06), 2197–2200 (2010)
10. Mehdi B., Mohammad B.: An overview to software architecture in intrusion detection system. Int. J. Soft Comput. Softw. Eng. (2012). doi:10.7321/jscse.v1.n1.1
11. Muna, M., Jawhar, T., Monica, M.: Design network intrusion system using hybrid fuzzy neural network. Int. J. Comput. Sci. Secur. **4**(3), 285–294 (2009)
12. Jakir, H., Rahman, A., Sayeed, S., Samsuddin, K., Rokhani, F.: A modified hybrid fuzzy clustering algorithm for data partitions. Aust. J. Basic Appl. Sci. **5**, 674–681 (2011)
13. Suguna, J., Selvi, A.M.: Ensemble fuzzy clustering for mixed numeric and categorical data. Int. J. Comput. Appl. **42**, 19–23 (2012). doi:10.5120/5673-7705
14. Vladimir, V.: The Nature of Statistical Learning Theory, 2nd edn., p. 188. Springer, New York (1995). ISBN-10: 0387945598
15. John, G.H.: Estimating continuous distributions in bayesian classifiers. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, (UAI' 95), pp. 338–345. Morgan Kaufmann Publishers Inc., San Francisco (1995)
16. Sang-Jun, H., Sung-Bae, C.: Evolutionary neural networks for anomaly detection based on the behavior of a program. IEEE Trans. Syst. Man Cybern. **36**, 559–570 (2005) doi:10.1109/TSMCB.2005.860136
17. Mehdi, M., Mohammad, Z.: A neural network based system for intrusion detection and classification of attacks. In: IEEE International Conference on Advances in Intelligent Systems - Theory and Applications (2004)
18. Zhou, T.-J.: The research of intrusion detection based on genetic neural network. In: Proceedings of the 2008 International Conference on Wavelet Analysis and Pattern Recognition, pp. 276–281, 30–31 Aug 2008. IEEE Xplore Press, Hong Kong (2008). doi:10.1109/ICWAPR.2008.4635789
19. Novikov, D., Yampolskiy, R.V., Reznik, L.: Anomaly detection based intrusion detection. In: Proceedings of the Third International Conference on Information Technology: New Generations, pp. 420–425, 10–12 April 2006. IEEE Xplore Press, Las Vegas (2006) doi:10.1109/ITNG.2006.33
20. Dahlia, A., Zainaddin, A., Mohd Hanapi, Z.: Hybrid of fuzzy clustering neural network over nsl dataset for intrusion detection system. J. Comput. Sci. **9**(3), 391–403 (2013). ISSN: 1549-3636 2013. doi:10.3844/jcssp.2013391 403 [Science Publications]
21. Tartakovskya, A.G., Rozovskii, B.L., Rudolf, B., Blazek, R.B., Kim, H.J.: A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. IEEE Trans. Signal Process. **54**(9) (2006). doi:10.1109/TSP.2006.879308
22. Mukhopadhyay, I.: Implementation of Kalman filter in intrusion detection system. In: Proceeding of ISCI Technologies, Vientiane (2008)
23. Simei Gomes, W., Lubica, B., Kasabov Nikola, K.: Adaptive learning procedure for a network of spiking neurons and visual pattern recognition. In: Advanced Concepts for Intelligent Vision Systems. Springer, New York (2006)
24. Babar, K., Khalid, F.: Generic unpacking techniques., Computer, Control and Communication, 2nd International Conference on IC4 IEEE (2009), DOI:10.1109/IC4.2009.4909168 (2009)

25. Royal, P., Halpin, M., Dagon, D., Edmonds, R.: Polyunpack: automating the hidden-code extraction of unpack-executing malware. In: ACSAC (2006)
26. Kang, M., Poosankam, P., Yin, H.: Renovo: a hidden code extractor for packed executables. In: 2007 ACM Workshop on Recurring Malcode (2007)
27. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: fast, generic, and safe unpacking of malware. In: Proceedings of the ACSAC, pp. 431/441 (2007)
28. Yegneswaran, V., Saidi, H., Porras, P., Sharif, M.: Eureka: a framework for enabling static analysis on malware. Technical Report SRI-CSL-08-01 (2008)
29. Danielescu, A.: Anti-debugging and anti-emulation techniques. Code-Breakers J. **5**(1), 27–30 (2008)
30. Farooq, M.: PE-Miner: mining structural information to detect malicious executables in realtime. In: 12th Symposium on Recent Advances in ID, pp. 121–141. Springer, New York (2009)
31. Shaq, M., Tabish, S., Farooq, M.: PE-probe: leveraging packer detection and structural information to detect malicious portable executables. In: Proceedings of the Virus Bulletin Conference (2009)
32. 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
33. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. ML Res. **7**, 2721–2744 (2006)
34. Ugarte-Pedrero, X., Santos, I., Bringas, P.G., Gastesi , M., Esparza, J.M.: Semi-supervised Learning for Packed Executable Detection, Network and System Security (NSS), 5th International Conference on, (2011). DOI: 10.1109/ICNSS.2011.6060027
35. Ugarte-Pedrero, X., Santos, I., Laorden, C., Sanz, B., Bringas, G.P.: Collective classification for packed executable identification. In: ACM CEAS (2011)
36. 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
37. Ye, Y., Wang, D., Li, T., Ye, D.: Imds: Intelligent Malware Detection System. ACM, New York (2007)
38. Chandrasekaran, M., Vidyaraman, V., Upadhyaya S.J.: Spycon: emulating user activities to detect evasive spyware. Performance, Computing, and Communications Conference, 2007. In: IPCCC 2007. IEEE International Conference on (2007). DOI:10.1109/PCCC.2007.358933
39. 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, 2008, MALWARE 2008, pp. 77–84 (2008)
40. Stamp, M., Attaluri, S., McGhee, S.: Profile hidden marko v models and metamorphic virus detection. J. Comput. Virol. **5**(2):151-169 (2009). DOI: 10.1007/s11416-008-0105-1
41. Santamarta, R.: Generic detection and classification of polymorphic malware using neural pattern recognition, white paper, ReverseMode. http://www.reversemode.com/ (2006)
42. Yoo, I.: Visualizing windows executable viruses using self-organizing maps. In: VizSEC/DMSEC '04: ACM Workshop (2004)
43. Livshits, V.B., Lam, M.S.: Finding Security vulnerability in Java applications with static analysis. In: Proceedings of the 14th USS, August 2005
44. Halfond, W.G.J., Orso, A., Manolios, P.: WASP: protecting web applications using positive tainting and syntax-aware evaluation. IEEE Trans. Softw. Eng. **34**, 181–191 (2008)
45. Buehrer, G.T., Weide, B.W., Sivilotti, Using Parse tree validation to prevent SQL injection attacks. In: Proceeding of the 5th International Workshop on Software Engineering and Middleware (SEM '056), pp. 106–113, September 2005
46. Cova, M., Balzarotti, D., Felmetsger, V., Vigna, G.: Swaddler: an approach for the anamoly based character distribution models in the detection of SQL injection attacks. In: Recent Advances in Intrusion Detection System, pp. 63–86. Springerlink, New York (2007)

47. Gerstenberger, R.: Anomaliebasierte Angriffserkennung im FTP-Protokoll. Master's Thesis, University of Potsdam, Germany (2008)
48. Dùssel, P., Gehl, C., Laskov, P., Rieck, K.: Incorporation of application layer protocol syntax into anomaly detection. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 188–202. Springer, Heidelberg (2008)
49. Bockermann, C., Apel, M., Meier, M.: Learning sql. for database intrusion detection using context-sensitive modelling. In: Detection of Intrusions and Malware, and Vulnerability Assessment, vol. 5587/2009, pp. 196–205. Springer Berlin/Heidelberg (2009)
50. Dewhurst, R.: Damn Vulnerable Web Application (DVWA). http://www.dvwa.co.uk/ (2012)
51. Bernardo Damele, A.G., Stampar, M.: Sqlmap: automatic SQL injection and database takeover tool. http://sqlmap.sourceforge.net/ (2012)
52. Valeur, F., Mutz, D., Vigna, G.: A Learning-based approach to the detection of SQL attacks. In: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, Vienna, pp. 123–140 (2005)
53. Wang, Y., Li, Z.: SQL injection detection with composite kernel in support vector machine. Int. J. Secur. Appl. **6**(2), 191 (2012)
54. Romi Rawat, R., Kumar Shrivastav, S.: SQL injection attack detection using SVM. Int. J. Comput. Appl. **42**(13), 0975–8887 (2012)
55. Huang, Z., Hong Cheon, E.: An approach to prevention of SQL injection attack based on machine learning. In: Proceedings of the First Yellow Sea International Conference on Ubiquitous Computing, Weihai (2011)
56. Hong Cheon, E., Huang, Z., Sik Lee, Y.: Preventing SQL injection attack based on machine learning. Int. J. Adv. Comput. Technol. **5**(9), (2013). doi:10.4156/ijact.vol5.issue9.115
57. Thorpe, S.J., Arnaud, D., van Rullen, R.: Spike-based strategies for rapid processing. Neural Netw. **14**(6–7), 715–725 (2001)
58. 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. Neurocomputing **38–40**(1–4), 539–545 (2000)
59. Thorpe, S.J., Gautrais, J.: Rank order coding. In: CNS '97: Proceding of the 6th Annual Conference on Computational Neuroscience: Trends in Research, pp. 113–118. Plenum Press, New York (1998)
60. Nikola, K.: Evolving Connectionist Systems: The Knowledge Engineering Approach. Springer, New York (2006)
61. Schliebs, S., Defoin-Platel, M., Kasabov, N.: Integrated feature and parameter optimization for an evolving spiking neural network. In: 15th International Conference, ICONIP 2008. Lecture Notes in Computer Science, vol. 5506, pp. 1229–1236, 25–28 Nov 2008. Springer, New York (2009)
62. Shrivastava, S., Singh, M.P.: Performance evaluation of feed-forward neural network with soft computing techniques for hand written English alphabets. Appl. Soft Comput. **11**(1), 1156–1182 (2011)
63. Shao, Y.E., Hsu, B.-S.: Determining the contributors for a multivariate SPC chart signal using artificial neural networks and support vector machine. J. ICIC **5**(12(B)), 4899–4906 (2009)
64. Chou, P.-H., Hsu, C.-H., Wu, C.-F., Li, P.-H., Wu, M.-J.: Application of back-propagation neural network for e-commerce customers patterning. ICIC Express Lett. **3**(3(B)), 775–785 (2009)
65. He, C., Li, H., Wang, B., Yu, W., Liang, X.: Prediction of compressive yield load for metal hollow sphere with crack based on artificial neural network. ICIC Express Lett. **3**(4(B)), 1263–1268 (2009)
66. Wu, J.K., Kang, J., Chen, M.H., Chen, G.T.: Fuzzy neural network model based on particle swarm optimization for short-term load forecasting. In: Proceedings of CSU-EPSA **19**(1), 63–67 (2007)
67. Li, D.K., Zhang, H.X., Li, S.A.: Development cost estimation of aircraft frame based on BP neural networks. FCCC **31**(9), 27–29 (2006)

68. Karimi, B., Menhaj, M.B., Saboori, I.: Multilayer feed forward neural networks for controlling decentralized large-scale non-affine nonlinear systems with guaranteed stability. Int. J. Innov. Comput. Inf. Control **6**(11), 4825–4841 (2010)
69. ZareNezhad, B., Aminian, A.: A multi-layer feed forward neural network model for accurate prediction of fue gas sulfuric acid dew points in process industries. Appl. Therm. Eng. **30**(6–7), 692–696 (2010)
70. Huang, L., Song, Q., Kasabov, N.: Evolving connectionist system based role allocation for robotic soccer. Playing, Intelligent Control, 2005. Proceedings of the IEEE International Symposium on (2005). Mediterrean Conference on Control and Automation (2005). DOI:10.1109/.2005.1466988
71. Kasabov, N.: Evolving fuzzy neural networks for on-line supervised/ unsupervised, knowledge–based learning. IEEE Trans. Cybern. **31**(6), 902–918 (2001)
72. Song, Q., Kasabov, N.: Weighted data normalization and feature selection. In: Proceedings 8th Intelligence Information Systems Conference (2003)
73. Kasabov, N., Song Q.: GA-parameter optimization of evolving connectionist systems for classification and a case study from bioinformatics. In: 9th Conference on Neural Information ICONIP '02, IEEE ICONIP. 1198128 (2002)
74. Vlassis, N.: A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Morgan and Claypool Publishers, San Rafael (2008). ISBN: 978-1-59829-526-9
75. Stolfo Salvatore, J., Wei, F., Lee, W., Andreas, P., Chan, P.K.: Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection: results from the JAM project. In: Proceedings of DARPA Information Survivability Conference and Exposition, DISCEX '00 (2000)
76. Jeff, H.: Introduction to Neural Networks with Java, 1st edn. (2008). ISBN: 097732060X
77. Goh, L., Song, Q., Kasabov, N.: A novel feature selection method to improve classification of gene expression data. In: 2nd Asia-Pacific IT Conference, vol. 29 (2004)
78. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
79. Zwillinger, D., Kokoska, S.: CRC Standard Probability and Statistics Tables and Formulae, CRC Press Print (1999). ISBN: 978-1-58488-059-2, eBook ISBN: 978-1-4200-5026-4
80. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: J. Artif. Intell. Res., **16**(1), 321–357 (2002)