TESTABILITY OF INFORMATION LEAK IN THE SOURCE CODE FOR INDEPENDENT TEST ORGANIZATION BY USING BACK PROPAGATION ALGORITHM

Al-Khanjari, Z.
Alani, A.
Department of Computer Science, College of Science,
Sultan Qaboos University, Muscat, Oman

Abstract

A strategy for software testing integrates the design of software test cases into a well-planned series of steps that results in a successful development of the software security. The strategy provides the secure source code test by Independent Test Organization (ITO) that describes the steps to be taken, when, and how much effort, time, and resources will be required. The strategy incorporates test planning, test case design, test execution, test result collection and test leak information and evaluation. In this work we speak about the testability of leak information in source code and how to detect and protect it inside the ITO. In this paper we present a privacy preserving algorithm for the neural network learning to detect and protect the leak information in source code between two parties the programmer (source code) and Independent Test Organization (Sensor). We show that our algorithm is very secure and the sensor inside Independent Test Organization is able to detect and protect all leaks information inside the source code. We demonstrate the efficiency of our algorithm by experiments on real world data. We present new technology for software Security using Back Propagation algorithm. That is embedded sensor to analyze the source code inside the ITO. By using embedded sensor we can detect and protect in real time all the attacks or leaks of information inside the source code. The connection between an Artificial Neural Networks and source code analysis inside Independent Test Organization is providing a great help for the software security.

Keywords: Software Security, Artificial Neural Networks, Back Propagation, Independent Test Organization, Testability

Introduction

Building a secure channel in source code is one of the most challenging areas of research and development in modern communication for software security. Attacks on source code infrastructures and software computer are becoming an increasingly serious problem nowadays [1]. Therefore, several information security techniques are available today to protect information systems against unauthorized use, duplication, alteration, destruction and viruses attack. Vapnik [2] applied a supervised learning algorithm based on the pioneering work. Joachims [3] stated that statistical learning theory have been successfully applied in a number of classification problems.Ghosh [4] Applied machine learning algorithms in anomaly detection. This had also received considerable attention.Honig and colleagues [5] described Adaptive Model Generation (AMG) as a real-time architecture for implementing data-mining-based intrusion detection systems. AMG uses SVMs as one specific type of model-generation algorithms for anomaly detection. Mukkamala and colleagues [6] compared the performance of neural network-based and SVM-based systems for intrusion detection using a set of DARPA benchmark data. If labeled data is available and used as input to a supervised network, an output representing the classes can be produced [7]. This type of system is limited to the classifications present in the training data. IDS rules are used as a basis for network anomaly detection reporting in the HISA algorithm [8]. Mohd and colleagues [9] provided a roadmap to industry personnel and researchers to assess, and preferably, quantify software testability in design phase. Li and colleagues [10] proposed an anomaly based network intrusion detection system based on Multilayer perceptron with a single hidden layer trained by Back Propagation learning algorithm. The system operation was divided into three stages: Input Data Collection and Preprocessing, Training, and Detection stage. The result for the proposed module was 95% detection rate. Agarwal and Agarwal [11] stated that the connection between an Artificial Neural Networks and cryptography is providing a great help for the security concerns. Singh and Ramkumar [12] presented a new technology for Security reasons. This is represented by Robots and embedded systems using Camera inside the devices. The authors of this paper used Back propagation algorithm to detect the face in a proper manner and a right direction without any errors and transferred images into memories in micro controller chip. Kemerlis and colleagues [13] suggested to employ system tracing facilities and data indexing services, and combine them in a novel way to detect data leaks. Chothia and Guha [14] presented a statistical test for detecting information leaks in systems with continuous outputs. Chothia and Guha also used continuous mutual information to detect the information leakage from trial runs of a probabilistic system. The main contributions of this paper are:

- show how to detect the leak information in source code which can be used to measure information leakage by using Back propagation Neural Network algorithm.
- test for the presence of information leak and detect by the Independent Test Organization.
- use this test to find, detect and fix any information leak in the source code from the programmers or use the trap door to remote access to the software.

In this paper, we use the supervisor learning in Back propagation Neural Network algorithm to find and detect information leak if any. The level of the error must be zero or less than 0.000001. The rest of the paper is organized as follows. Section 2 explains the work of Independent Test Organization (ITO) and what is the purpose of the ITO? What are the different approaches to keep test costs under control? Section 3 discusses the Artificial Neural Network (ANN), its design and usage. Also, this section explains the algorithm of Back propagation Neural Network and the procedure of learning. Section 4 discusses the development of a test for information leakage in source code. Section 5 provides concluding remarks of the work.

Independent Test Organization (ITO)

An Independent Test Organization is an organization, a person, or a company that tests products, materials, software, etc, according to agreed requirements. The test organization can be affiliated with the government, universities or can be an independent testing laboratory. They are independent because they are not affiliated with the producer nor the user of the item being tested: no commercial bias is present. These "contract testing" facilities are sometimes called "third party" testing or evaluation facilities [15]. An Independent Test Organization might also be an organization that tests application according to standard requirements. Test organizations specialize in testing and are majorly independent of the supplier of application and the company that purchases the application. Testing is a very important aspect of any application to perform its functional and non-functional behavior and whether it behaves as per business objective. An unsuccessful testing, project may allow a substandard application to go live. This might be the reputation of the organization [16]. Independent testing might have a variety of purposes, such as:

1. Verifying if the requirements of a specification, regulation, or contract are met.
2. Deciding if a new product development program is on track: Demonstrate proof of concept.
3. Providing standard data for other scientific, engineering, and quality assurance functions.
4. Validating suitability for end-use.
5. Providing a basis for technical communication.
6. Providing a technical means of comparison of several options.
7. Providing evidence in legal proceedings: forensics, product liability, patents, product claims, etc.
8. Solving problems with current products or services.
9. Identifying potential cost savings in products or services.

Software testability is the tendency of code to reveal existing faults or information leak.

This paper proposes the use of a software testability to detect and protect the information leak inside ITO throughout the development process by using Back Propagation Algorithm. We further believe that software testability analysis to detect the leak of information in real time can play a crucial role in quantifying the likelihood that faults are not hiding after finishing the testing process, which does not result in any failures for the current version. Testability is one of the major factors determining the time and effort needed to test software system. It is costly to redesign a system during implementation or maintenance in order to overcome the lack of testability [17]. There are different approaches to keep test costs under control and to increase the quality of the product under test [18] as shown in Figure 1.

1. improve the software specification and documentation,
2. reduce or change functional requirements to ease testing,
3. use better testing techniques,
4. use better testing tools,
5. improve the testing process,
6. train people, and
7. improve the software design and implementation.

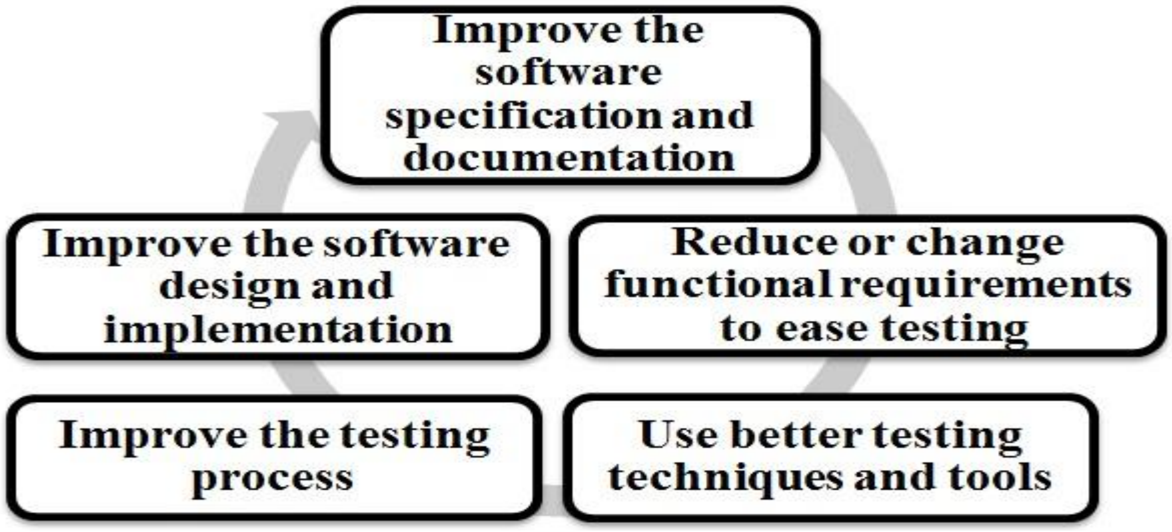
Improve the software specification and documentation
Reduce or change functional requirements to ease testing
Use better testing techniques and tools
Improve the testing process
Improve the software design and implementation

Figure 1: Steps for Increasing the Quality of the Product Under Test

3. Artificial Neural Network (ANN)

Biologists have studied biological neural networks for many years. The human brain looks like a network. Discovering how the brain works has been an ongoing effort that started more than 2000 years ago. Information about the function the brain was accumulated, a new technology emerged as the quest for an “Artificial Neural Network” start. The brain processes information super quickly and super accurately. It can be trained to recognize patterns and to identify incomplete patterns [19]. While designing ANN we should be concerned with the following:

1. Network topology
2. Number of layers in the network
3. Number of neurons or nodes
4. Learning algorithm to be adopted
5. Network performance
6. Degree of adaptability of the ANN (i.e. to what extent the ANN is able to adapt itself after training).

A Neural network’s ability to perform computations is based on the hope that we can reproduce some of the flexibility and power of the human brain by artificial means. Network computation is performed by a dense mesh of computing nodes and connections. They operate collectively and simultaneously on most or all data inputs. The basic processing elements of neural networks are called artificial neurons, or simply neurons [19]. Therefore, neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths and processing performed at computing elements or nodes [20].

Back Propagation Neural Network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems process information. It is configured for a specific application through a specific learning process. The most commonly used family of neural networks for pattern classification tasks is the feed-forward network, which includes multilayer perceptron and Radial-Basis Function (RBF) networks.

Back Propagation is a feed forward supervised learning network. The general idea with the back propagation algorithm is to use gradient descent, to update the weights and to minimize the squared error between the network output values and the target output values. The update rules are derived by taking the partial derivative of the error function with respect to the weights to determine each weight’s contribution to the error. Then, each weight is adjusted. This process occurs iteratively for each layer of the network. The concept of the process is to start with the last set of weights, and work back towards the input layer. This concept is named as “Back Propagation”.

The network is trained to perform its ability to respond correctly to the input patterns that are used for training. Also, to provide good response to input that are similar. Propagation analysis is the process concerned with determination of the probability that a forced change in an internal computational state causes a change in the program’s output. In other words, it is the probability that an error in the data state at a location causes an output error for a given input distribution. Propagation of a data state error occurs when the output is affected by the data state.

Propagation analysis involves three things:

1. Obtaining a data state at a location in the code.
2. Perturbing the data state.
3. Executing the code to completion and examining the resulting output to see if the perturbed data state has changed the output.

Propagation analysis is similar to a strong mutation testing in that the results at the end of the execution of the original program are compared with the results obtained when a data state is corrupted [21]. The algorithm Works as shown in Table 1 [22]:

Table 1: Back Propagation Neural Network Algorithm

1. **Apply the inputs to the network and work out the output – remember this initial output could be anything, as the initial weights were random numbers.**
2. **Work out the error for neuron B. The error is *what you want? What you actually get?* in other words:**
   Error ^{B} = Output ^{B}*(1-Output ^{B})*(Target ^{B} – Output ^{B})
   The "Output*(1-Output)" term is necessary in the equation because of the Sigmoid Function – if we only were using a threshold neuron it would just be (Target –Output).
3. **Change the weight. Let W^{+AB} be the new (trained) weight and W^{AB} be the initial weight.**
   W^{+AB} = W^{AB} + (Error ^{B} x Output ^{A})
   Notice that it is the output of the connecting neuron (neuron ^{A}) we use (not ^{B}). We update all the weights in the output layer in this way.
4. **Calculate the Errors for the hidden layer neurons. Unlike the output layer we can't calculate these directly because we don't have a Target, so we *Back Propagate* them from the output layer. Hence the name of the algorithm. This is done by taking the Errors from the output neurons and running them back through the weights to get the hidden layer errors. For example if neuron A is connected to B and C then we take the errors from B and C to generate an error for A.**
   Error ^{A} = Output ^{A}*(1 - Output ^{A})*(Error ^{B} + Error ^{C})
   Again, the factor "*Output*(1 - Output)*" is present because of the sigmoid squashing function.
5. **Having obtained the Error for the hidden layer neurons now proceed as in stage three to change the hidden layer weights. By repeating this method we can train a network of any number of layers.**

After choosing the weights of the network randomly, the back propagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

1. Feed-forward computation
2. Back propagation to the output layer
3. Back propagation to the hidden layer
4. Weight updates

The algorithm is stopped when the value of the error function has become sufficiently small.

Back propagating – Learning

The network is first initialized by setting up all its weights to be small random numbers between –1 and +1. The input pattern is applied and the output calculated (this is called the forward pass). The calculation gives an output which is completely different to what you want (the Target), since all the weights are random. We then calculate the Error of each neuron, which is essentially: Target – Actual Output (i.e. what you want – What you actually get). This error is then used mathematically to change the weights in such a way that the error will get smaller.

In other words, the Output of each neuron will get closer to its Target (this part is called the reverse pass). The process is repeated again and again until the error is minimal.

Learning Procedure

1. Randomly assign weights (between +1 and -1)
2. Present inputs from training data, propagate to outputs

3. Compute outputs O; adjust weights according to the delta rule, back propagate the errors. The weights will be nudged closer so that the network learns to give the desired output.
4. Repeat; stop when no errors, or less than 0.000001.

Detect Leakage Information

The concept of the brain as a computer has been part of the modern scientific. This path has led to new fields of research including artificial intelligence and neural networks. Connections between computation and the brain have been studied extensively using Artificial Neural Networks (ANN) and inspired by biological neural networks. One aspect of the complexity of nervous systems is their intricate morphology, particularly the interconnectivity of their neuronal processing elements. Synapses are the ends of the connections within the nervous system [23].

There are two types of synapses:

1. Chemical synapses, which use neurotransmitters,
2. Electrical synapses, which provide direct electrical coupling to the synapsed cell. Neurons are very polarized cells, with long and thin extensions.

ANNs consist of ‘neuron’ nodes connected by ‘synapses’ of variable strength. They can be trained to perform a given task through algorithmic modification of the synaptic weights. A desired input-output relationship can be generated for a known set of examples, after which the ANN can be used to process unknown inputs.

In this paper we focus on source code as our measure of information leakage. We describe how it can be calculated and learned to detect the information leakage. There are two main obstacles to detecting the leak of information of a real system in the source code:

1. We must find random numbers assigned for weights that reflect the source code under test. In order to detect the leakage through present inputs from training data: propagate to outputs, compute outputs O, adjust weights according to the delta rule and back propagate the errors. The weights will be nudged closer so that the network learns to give the desired output
2. We must calculate the error when learning and stop it when the leakage is zero or less than 0.000001. This is the biggest challenge to discover the accuracy of information leaking in the source code. This works as embedded sensor inside the source code to detect the leakage of information. Figure 2 shows Weights of Network Back Propagation.

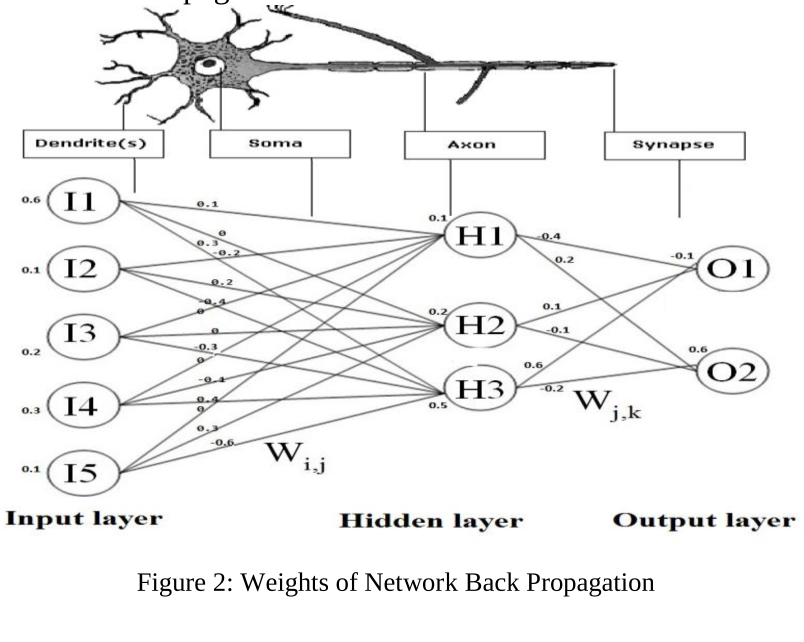

Figure 2: Weights of Network Back Propagation

We could stop it once the network can recognize all the letters successfully, but in practice it is usual to let the error fall to a lower value first. This ensures that the letters are all being well recognized. You can evaluate the total error of the network by adding up all the errors for each individual neuron and then for each pattern in turn to give you a total error as shown in Figure 3.

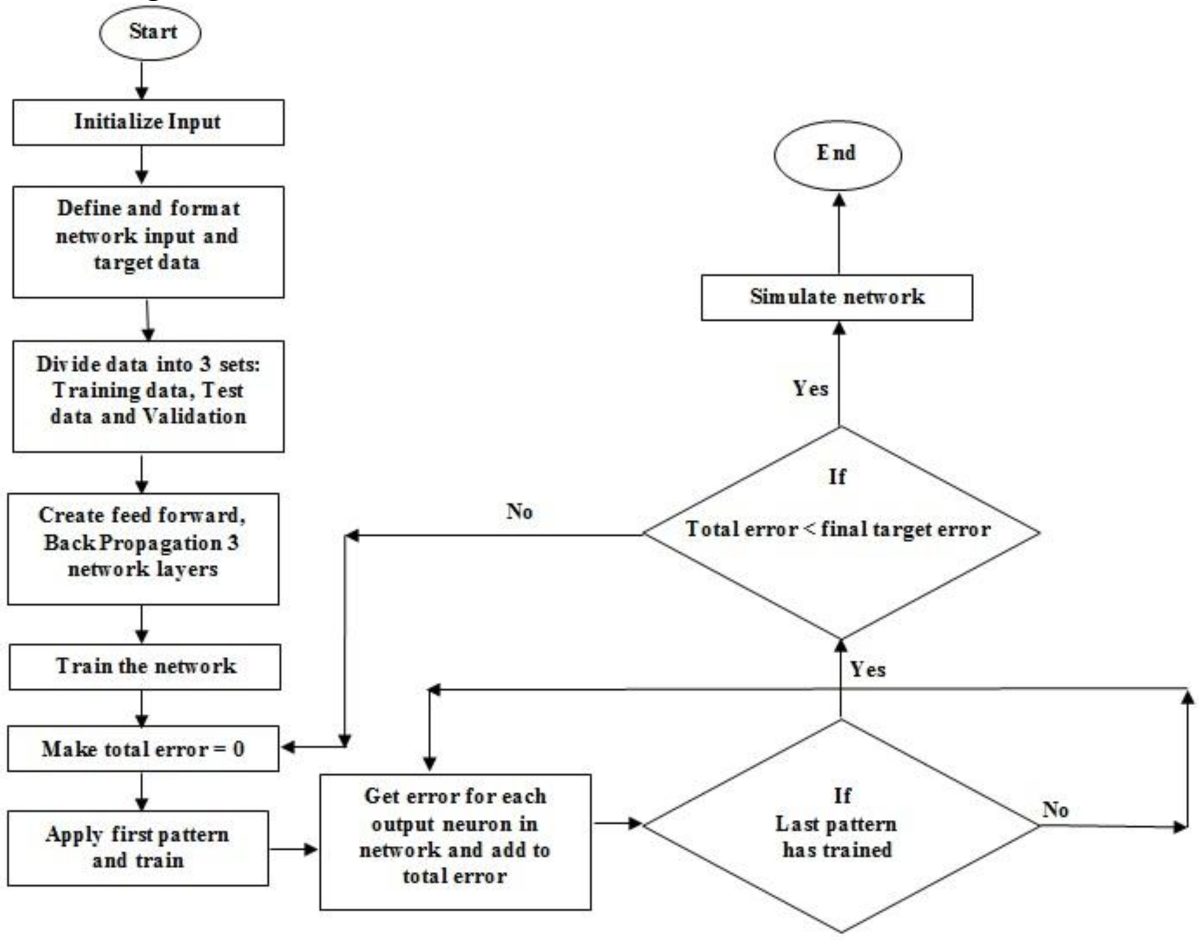
Start
Initialize Input
Define and format network input and target data
Divide data into 3 sets: Training data, Test data and Validation
Create feed forward, Back Propagation 3 network layers
Train the network
Make total error = 0
Apply first pattern and train
Get error for each output neuron in network and add to total error
If Last pattern has trained
No
Yes
If Total error < final target error
No
Yes
Simulate network
End

Figure 3: Artificial Neural Network Flowchart

Forward Pass for Training and Detection of the Word (Class) in the Source Code
Activations of the Hidden Layers

An important special case of feed-forward networks is the layered networks with one or more hidden layers. We give explicit formulas for the weight updates and show how they can be calculated using linear algebraic operations. We also show how to label each node with the back propagated error in order to avoid redundant computations.

netH1= I1 *wI1H1+ I2* wI2H1+ I3 *wI3H1+ I4* wI4H1+bH1+bH1
=0.6*0.1 + 0.1* (-0.2) + 0.2*0 + 0.3*0 + 0.1*0 + 0.1=0.14
oH1=1/ (1+e- netH1) =0.53
netH2= I1 *wI1H2+ I2* wI2H2+ I3 *wI3H2+ I4* wI4H2+bH2+bH2
=0.6 *0 + 0.1*0.2 + 0.2*0 +0.3*(-0.1) + 0.1*0.3 + 0.2=0.22
oH2=1/ (1+e- netH2) =0.55
netH3= I1 *wI1H3+ I2* wI2H3+ I3 *wI3H3+ I4* wI4H3+bH3+bH3
=0.6*0.3 + 0.1*(-0.4) + 0.2*(-0.3) + 0.3*.0.4 + 0.1* (-0.6) + 0.5=0.64
oH3=1/ (1+e- netH3) =0.65

Activations of the Output Layers

It is important not to update any weights until all errors have been calculated. It is easy to forget this and if new weights were used while calculating errors, results would not be valid. Here, a quick second pass using new weights is needed to see if error has decreased. The vector O is presented to the network. The vectors O(1) and O(2) are computed and stored. The evaluated derivatives of the activation functions are also stored at each unit.

netO1=oH1*wH1O1+ oH2*wH2O1 + oH3*wH3O1 +bO1
=0.53*(-0.4) + 0.55* 0.1 + 0.65* 0.6+ (-0.1) =0.13
oO1=1/ (1+e- netO1) =0.53
netO2=oH1*wH1O2+ oH2*wH2O2 + oH3*wH3O2 +bO2
=0.53*0.2+0.55*(-0.1) +0.65*(-0.2) +0.6=0.52
oO2=1/ (1+e- netO2) =0.63

Backward Pass for Detecting Word (Class) in the Source Code

Calculate the Output Errors: errerO1 and errerO2 (note that doO1=1, doO2=0)

Now errors have to be propagated from the hidden layer down to the input layer. This is a bit more complicated than propagating error from the output to the hidden layer.

errerO1 = (doO1- oO1) * oO1 * (1- oO1) = (1-0.53)*0.53*(1-0.53) = 0.12
errerO2 = (doO2- oO2) * oO2 * (1- oO2) = (0-0.63)*0.63*(1-0.63) = -0.15

Calculate the New Weights Between the Hidden and Output Layers (η=0.1)

It is important not to update any weights until all errors have been calculated. It is easy to forget this and if new weights were used while calculating errors, results would not be valid. Here, a quick second pass using new weights is needed to see if error has decreased. After computing all partial derivatives, the network weights are updated in the negative gradient direction. Learning constant defines the step length of the correction.

ΔwH1O1= η * errerO1 * oO1 = 0.1*0.12*0.53=0.006
$wH1O1^{new}$ = $wH1O1^{old}$ + ΔwH1O1 = -0.4+0.006= -0.394
ΔwH1O2= η * errerO2 * oO1 = 0.1*-0.15*0.53=-0.008
$wH1O2^{new}$ = $wH1O2^{old}$ + wH1O2 = 0.2-0.008=-0.19

Similarly for $wH2O1^{new}$, $wH2O2^{new}$, $wH3O1^{new}$ and $wH3O2^{new}$

For the biases bO1 and bO2 (remember: biases are weights with input 1):

ΔbO1= η * errerO1 * 1 = 0.1*0.12=0.012
$bO1^{new}$ = $bO1^{old}$ + ΔbO1 = -0.1+0.012=-0.012

Similarly for bO2

Calculate the Errors of the Hidden Layers: errerH1, errerH2 and errerH3

In the case of error > 1 input - output patterns, an extended network is used to compute the error function for each of them separately. The weight corrections are computed for each pattern and so we get the corrections

errerH1 = oH1 * (1- oH1) * (wH1O1* errerO1 + wH1O2 * errerO2)
= 0.53*(1-0.53) (-0.4*0.12+0.2*(-0.15)) = -0.019

Similarly for errerH2 and errerH3

Calculate the New Weights Between the Input and Hidden Layers (η=0.1)

We speak of batch or off-line updates when the weight corrections are made in this way. Often, however, the weight updates are made sequentially after each pattern presentation (this is called on-line training). In this case, the corrections do not exactly follow the negative gradient direction. However, if the training patterns are selected randomly the search direction oscillates around the exact gradient direction. On average, the algorithm implements a form of descent in the error function. The rationale for using on-line training is that adding some noise to the gradient direction can help to avoid falling into shallow local minima of the error function. Also, when the training set consists of thousands of training patterns, it is very expensive to compute the exact gradient direction since each epoch (one round of presentation of all patterns to the network) consists of many feed-forward passes and on-line training becomes more efficient.

$\Delta wI1H1 = \eta * errerH1 * I1 = 0.1*(-0.019)*0.6 = -0.0011$

$wI1H1^{new} = wI1H1^{old} + \Delta wI1H1 = 0.1-0.0011=0.0989$

Similarly for:

$wI2H1^{new}$, $wI3H1^{new}$, $wI4H1^{new}$, $wI5H1^{new}$, wI1H2new, $wI2H2^{new}$, $wI3H2^{new}$, $wI4H2^{new}$, $wI5H2^{new}$, $wI1H3^{new}$, $wI2H3^{new}$, $wI3H3^{new}$, $wI4H3^{new}$ and $wI5H3^{new}$; bH1, bH2 and bH3

Usefulness of Other Training

Repeat the procedure for the other training to detect the information leak in the source code such as class, mail, web, URL etc. The back propagated error can be computed in the same way for any number of hidden layers and the expression for the partial derivatives of E keeps the same analytic form.

Conclusion

In this paper was designed a method to detect information leakage in the source code to help independent test organization to detect the trap door of leak information.

The proposed system classifies an architecture based on learning using an enhanced resilient Back Propagation Neural Network algorithm. It is used to detect information leakage in the source code. The system is monitored by real time technology. It is able to extract leakage of information from the source code that designed with high detection, and accuracy and calculate the error when learning and stop it when the error is zero or less than 0.000001.

The following points are concluded from the proposed system.

1. The excellent detection rate for information leakage is very encouraging.
2. The sensors after learning have been the simplest in the cases where they embedded themselves in the source code and checked all attacks.
3. This detection of leak information can operate without any external components.
4. The prototype implemented is able to detect information leak.
5. Using automatic audit for detecting all information leakage provides high security to the independent test organization without using any other protection programs.
6. The proposed method to detect the leak of information using Java language is very flexible in dealing with any kind of operating systems.

By using the real time technique, we can use our method to detect the information leak in the source code which deals with them without returning to the programmer or getting the help of the owner of the source code.

References:

Abraham, A., Grosan, C. & Chen, Y. (2006). Evolution of Intrusion Detection Systems. [Online] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16 [Accessed on 18 October 2011].

Vapnik, V. (1995). The Nature of Statistical Learning Theory. Springer, New York, USA.

Joachims, T. (1998). Making large-scale support vector machine learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods: Support Vector Machines. MIT Press, Cambridge, MA,USA.

Ghosh, A., Schwartzbard, A. & Schatz, M. (1999). Learning program behavior profiles for intrusion detection. In ID’99: Proceedings of the 1st conference on Workshop on Intrusion Detection and Network Monitoring.
Honig, A., Howard, A., Eskin, E. & Stolfo, S. (2002). Adaptive model generation: An architecture for the deployment of data mining-based intrusion detection systems. Data Mining for Security Applications.
Mukkamala, S. , Janoski, G. & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. The Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002), Vol. 2, pp. 1702 – 1707, Honolulu, HI.
Khan, L., Awad, M., Thuraisingham, B. (2007). A new intrusion detection system using support vector machines and hierarchical clustering’, The VLDB Journal, Vol. 16, pp. 507-521, 2007.
Vollmer, T. & Manic, M. (2009). Human Interface for Cyber Security Anomaly Detection Systems, HSI2009, 2nd International Conference on Human Systems Interactions, Catania Italy, May 28-30, 2009.
Mohd, N., Khan, R. & Mustafa, K. (2010). Testability Estimation Framework, International Journal of Computer Applications (0975 – 8887), 2(5), pp. 9-14, June 2010.
Li, J., Zhang, G. & Gu, U. (2004). The Research and Implementation of Intelligent Intrusion Detection System Based on Artificial Neural Network. Proceedings of the Third International Conference on Machine Laming and Cybernetics, pp. 26-29, Shanghai.
Agarwal, N. & Agarwal, P. (2013). Use of Artificial Neural Network in the Field of Security, MIT International Journal of Computer Science & Information Technology, pp. 42–44 ISSN 2230-7621 ©MIT Publications, 3(1), January 2013.
Singh, R. & Ramkumar, E. (2013). Embedded Systems and Robotics that Improving Security Model with 2D and 3D of Face -Recognition Access Control System Using Neural Networks International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, 2(6), pp. 197-200, January 2013.
Kemerlis, V., Pappas, V., Portokalidis, G. & Keromytis, A. (2010). iLeak: A Lightweight System for Detecting Inadvertent Information Leaks, IEEE Conference Publications - Computer Network Defense (EC2ND), pp. 21 - 28, European Conference, 2010.
Chothia, T. & Guha, A. (2011) Statistical Test for Information Leaks Using Continuous Mutual Information, Computer Security Foundations Symposium (CSF), IEEE 24th, pp. 177 – 190, 2011.
Wikipedia (2014). Independent Test Organization [Online]
http://en.wikipedia.org/wiki/Independent_test_organization [Accessed on: 12 January 2014].
The Mobile Guru (2014). Why Independent Testing & QA Makes Sense [Online] http://harshalkharod.blog.com/2011/03/02/independent-testing-outsource-testing/ [Accessed on: 20 January 2014].
Khatri, S., Chhillar, R. & Singh, V. (2011). Improving the Testability of Object-oriented Software during Testing and Debugging Processes, International Journal of Computer Applications, (0975 – 8887), 35(11), pp. 24-35, December, 2011.
Jungmayr, S. (2002). Design for Testability, In Proceedings of CONQUEST 2002, pp. 57-64, September 18th-20th, 2002, Nuremberg, Germany,.
Ranjana, R. & Aziz-ur-Rahman, M. (2006). Multivariable System Security Using ANN in the Libraries: Designing and Development, The 4th International Convention CALIBER-2006, INFLIBNET Centre, 2-4 February 2006, Gulbarga.
Jojodia, S. & Kogan, B. (1990). Transaction Processing in Multilevel-Secure databases using Replicated Architecture, The Proceedings of the 1990 Symposium on Research in Security and Privacy, pp. 360-368, May 1990.

Al-Khanjari, Z.A. (2006). Partial Automation of Sensitivity Analysis by Mutant Schemata Approach, the International Arab Journal for Information Technology (IAJIT), 3(1), pp. 82-93, January, 2006.
Rojas, R. (1996). Neural Networks, Springer-Verlag, Berlin, 1996.
Thomas, A. & Kaltschmidt, C. (2014). Bio-inspired Neural Networks, Springer International Publishing Switzerland, 2014.