UTILIZING NEURAL NETWORKS FOR EFFECTIVE INTRUSION DETECTION

Martin Botha[a] & Rossouw von Solms[b]

[a]Port Elizabeth Technikon, South Africa
[b]Port Elizabeth Technikon, South Africa

[a]bothamar@saps.org.za, Department of Information Technology, PE Technikon, Private Bag X6011, Port Elizabeth 6000

[b]rossouw@petech.ac.za, Department of Information Technology, PE Technikon, Private Bag X6011, Port Elizabeth 6000

ABSTRACT

Computer security, and intrusion detection in particular, has become increasingly important in today's business environment, to help ensure safe and trusted commerce between business partners as well as effective organisational functioning. Various approaches to intrusion detection are currently being utilized, but unfortunately in practice these approaches are relatively ineffective. New ways and means must, therefore, continuously be researched and defined. This paper will propose a proactive and dynamic model, based on trend analysis, fuzzy logic and neural networks that could be utilized to minimise and control intrusion to an organisation's computer system. The model will be based on the assumption that each user is unique and leaves a unique footprint on a computer system when using it. A back-propagation neural network was trained to implement this idea.

KEY WORDS

Computer security; intrusion detection; intrusion detection systems; fuzzy logic; neural network; pattern recognition.

UTILIZING NEURAL NETWORKS FOR EFFECTIVE INTRUSION DETECTION

1 INTRODUCTION

Over the last few decades, information has become an organization's most precious asset and most things an organisation does, involves using information in some way or another (Peppard, 1993; Von Solms, 1993). Organisations have therefore become increasingly dependent on the rapid access and management of information since more information is being stored and processed on network-based computers than ever before. The increase in connectivity not only provides access to larger and varied resources of data more quickly than ever before, it also provides an access path to the data from virtually anywhere on the network-based environment (Seleznyov, 2001).

Also, during the same period, modern computing systems have become increasingly complex due to constant dynamic changes in configurations, software and usage patterns. This situation creates almost unlimited opportunities for malicious persons, who are using software applications' and operating systems' vulnerabilities to successfully penetrate a computer system illegally (Seleznyov, 2002).

Evidence of this statement can be seen in the 2001 CSI/FBI Computer Crime and Security Survey (Power, 2001). According to this source the annual financial losses brought about by security breaches in 2001 have increased by 277% when compared to the results from 1997. The "2002 Computer Crime and Security Survey" confirms this by stating that the threat from computer crime and other information security breaches continues unabated and that the financial toll is mounting (Richardson, 2002).

Intrusion detection, originally proposed by Dorothy Denning in 1987, is an approach to counter these kinds of attacks (Denning, 1987). Intrusion detection is implemented by an intrusion detection system and today there are many commercial intrusion detection systems available. Generally speaking, most of the commercial implementations are relative ineffective and insufficient, which gives rise to the need for research on more dynamic intrusion detection systems (Dowland, 2000).

The objective of this paper is to propose such dynamic intrusion detection system, with the emphasis on the neural network component. The neural network component will implement the neural approach, which is based on the assumption that each user is unique and leaves a unique footprint on a computer system when using it. If a user's footprint does not match his/her reference footprint based on normal system activities, the system administrator or security officer can be alerted to a possible security breach.

To address the mentioned objective, the paper will first provide an overview of intrusion detection systems (IDS). It will commence with a discussion on the most important definitions and components of intrusion detection systems, as well as shortcomings currently encountered with misuse intrusion detection systems. This section will conclude with a short discussion on how these shortcomings can be addressed.

The paper will also propose a model that can be used to combat intrusion attacks proactively and will commence with the identification of nine major components of the proposed model. Each component will then be discussed with specific reference to the neural engine component and the neural approach.

Finally, the paper will describe a working prototype for the neural engine component of the proposed model. It will commence with background information on the software used to develop this prototype, followed by a brief discussion on the implementation process for the prototype. This section will conclude with a brief discussion on the test results obtained during testing.

2 OVERVIEW OF INTRUSION DETECTION

The need for a more dynamic intrusion detection system was highlighted as one of the main research areas currently being undertaken in the field of computer security. In order to define such a dynamic intrusion detection system one has to investigate the terms intrusion detection, intrusion detection system and intrusion detection analysis approach.

Bace defines intrusion detection as the process of intelligently monitoring the events occurring in a computer system or network, analysing them for signs of violations of the security policy (Bace, 2000). The intrusion detection process is performed by an intrusion detection system and is defined as a software or a hardware product that monitors the events occurring in a computer system or network and analyses them for signs of intrusions, which are defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a host or network (Bace, 2000).

An intrusion detection system normally consists of three functional components namely:

- An information source that provides a stream of event records;
- An analysis engine that identifies signs of intrusions; and
- A response component that generates reactions based on the outcome of the analysis engine (Bace, 2000, p.27).

The first component of an intrusion detection system, also known as the event generator, is a data source. Data sources can be categorized into four categories namely (Bace, 2000):

- Host-based monitors: This monitor collects data from sources internal to a computer, usually at the operating system level;
- Network-based monitors: This monitor collects network packets;
- Application-based monitors: This monitor collects data from running applications; and
- Target-based monitors: This monitor uses cryptographic hash functions to detect alterations to system objects and then compares these alterations to a policy.

The second component of an intrusion detection system is known as the analysis engine. This component takes information from the data source and examines the data for symptoms of attacks or other policy violations. The analysis engine can use one or both of the following analysis approaches:

- Misuse detection: An analysis engine that implements this strategy will search for something defined to be “bad”. This type of detection engine detects intrusions that follow well-known patterns of attacks (or signatures) that exploit known software vulnerabilities (Kumar and Spafford, 1995;Ilgun and Kemmerer, 1995). The primary limitation of this approach is that it looks only for known weaknesses, and may not be of much use in detecting unknown future intrusions (Seleznyov, 2000).
- Anomaly detection: An anomaly based detection engine will search for something rare or unusual. They analyse system event streams, using statistical techniques to find patterns of activity that appear to be abnormal. The main problems of anomaly based intrusion detection systems are that they tend to be computationally expensive, because several metrics are often maintained that need to be updated against every

system activity and they may be gradually trained incorrectly to recognize an intrusive behaviour as normal in the future due to insufficient data (Seleznyov, 2000).

The analysis engine can perform real time or non-real time intrusion detection. Real time detection is also referred to as on-line detection. On-line systems are designed to detect intrusions while they are happening, thereby allowing for quicker response from the response component of the system. The main disadvantage associated with this technique is that the system is computationally very expensive because it requires continuous monitoring. Non-real time detection is also referred to as off-line detection. Off-line intrusion detection systems are running periodically by detecting intrusions after-the-fact, based on system logs. The main drawback of the off-line systems is that it can mainly act reactively to intrusion attacks.

The third component of an intrusion detection system is the response manager. In basic terms, the response manager will only act when inaccuracies (possible intrusion attacks) are found on the system, by informing someone or something in the form of a response. Intrusion detection responses can either be categorised as active or passive. In active responses the system pro-actively influences the attack, for example, by blocking a certain IP address. Passive responses merely report and log these attacks, usually to a database.

To summarise, a dynamic intrusion detection system can be defined as a system that will identify intrusion attacks in real-time, that will update itself over a period of time with a minimum of new rules required from vendors and that will not allow an intruder to train the system incorrectly. To develop such a dynamic intrusion detection system, one could utilize a hybrid intrusion detection system, which employs both anomaly and misuse analysis strategies. The anomaly strategy will be used to detect new or unknown attacks, or other scenarios of concern, while misuse strategy will be used to detect known attack signatures and protect the integrity of the anomaly strategy by ensuring that a patient adversary cannot gradually change behaviour patterns to re-train the anomaly detector to accept attack behaviour as normal. The system could also use the on-line detection technique with special emphasis on low computational overheads and a proactive approach as well as an active and passive response manager. This strategy will be discussed in more detail in the next section.

3 THE NEGPAIM MODEL

Having introduced the terms intrusion detection, intrusion detection system and intrusion detection analysis approach, it is now possible to explain the hybrid intrusion detection system. This system is implemented through a model called Next Generation Proactive Identification Model (NeGPAIM). NeGPAIM is an improved version of PAIM that was developed as a misuse based intrusion detection system (Botha et al, 2002). Figure 1 shows the various components of NeGPAIM.

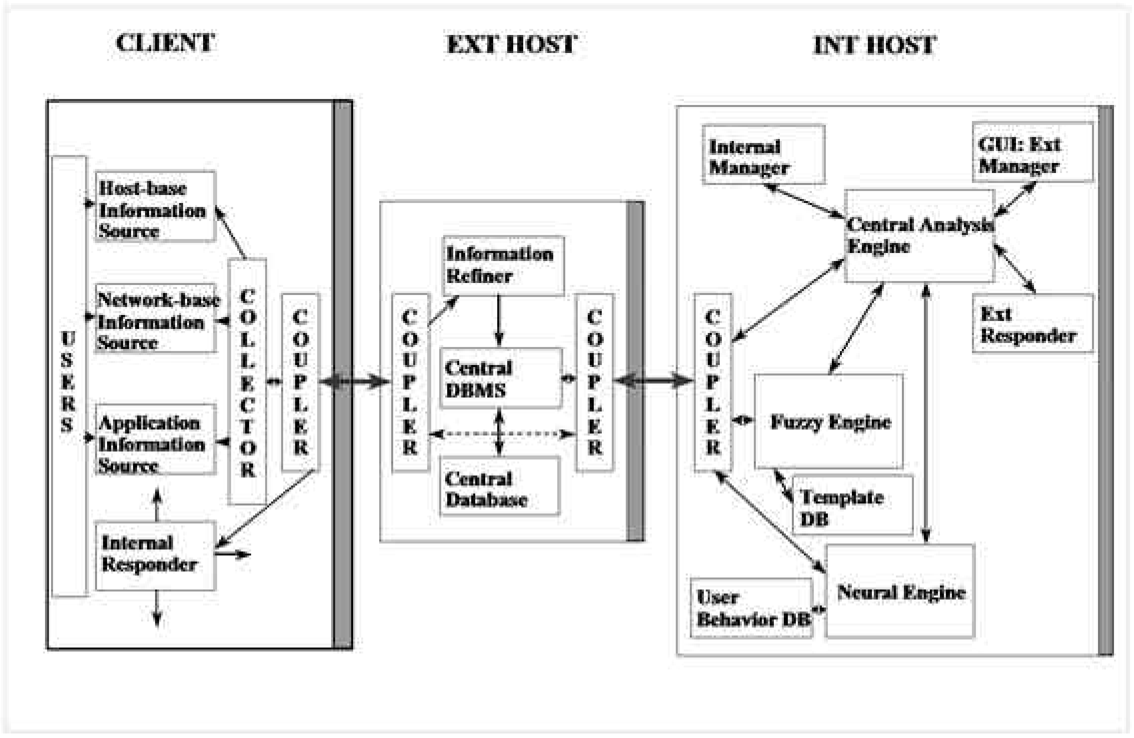


Figure 1: A General Representation of NeGPAIM

The different components of NeGPAIM are:

- Information Provider:
  Refers to the different sources that provide input data to the intrusion detection system.
- Collector:
  This component is a software unit that runs at the operating system level and is responsible for gathering the information from the information sources and sending it to the information refiner.
- Coupler:
  This is an interface that provides a way for the three tiers to communicate amongst themselves.
- Information Refiner:
  The refiner is responsible for structuring the data into a standard format so that it can then be used by the fuzzy engine and the neural engine.
- Fuzzy Engine:
  The fuzzy engine is one of the two low-level processing units of NeGPAIM and will process the input data. The engine implements the misuse detection approach and will firstly compute a template and a user action graph for each user and secondly map the two graphs to determine whether a user (intruder) is performing an intrusion attack. If the user

(intruder) is conducting an intrusion attack, the engine will notify the central analysis engine with an intrusion probability value. This process is performed on a continuous basis.

- Neural Engine:
  The neural engine is the second low-level processing unit and will also process the input data. This engine will process the data by searching for patterns of user behavior that appear to be abnormal. The engine will report any abnormal behavior to the central analysis engine by way of an intrusion probability value.
- Central Analysis Engine:
  The central analysis engine is a high level processing unit. The objective of this engine is to analyse the possibility that a user (intruder) is conducting an intrusion attack. The engine will achieve this by performing the following functions:

  X To calculate the statistical mean value (average) for both the probability values (fuzzy probability value and neural probability value) received from the low-level components for each user on a continuous basis;

  X To convert this statistical mean value (average) to an intrusion probability level that will be used to combat intrusion proactively and dynamically on a continuous basis; and

  X To ensure accurate intrusion identification by implementing threshold detection. The output of the central analysis engine is the generation of an alert signal that will be sent to the responder.
- Responder:
  The responder is responsible for taking the necessary action in the event of an intrusion.
- Manager:
  This component allows for the management and configuration of the intrusion detection system.

The nine major components, constituting the NeGPAIM model, were briefly introduced and discussed. The neural engine, the fuzzy engine and the central analysis engine are the main components of the model since they directly implement the misuse and anomaly approaches. See Botha et al, 2002 for a more detailed overview on NeGPAIM and Botha et al, 2001 for more detail on the fuzzy engine. The rest of the paper will only focus on the implementation of the neural engine.

4 THE NEURAL ENGINE

The previous section introduced NeGPAIM and its three main components. This section will discuss one of the three components namely the neural engine. The neural engine is based on the anomaly intrusion detection technique. The anomaly intrusion detection technique involves a process of establishing profiles of normal user behaviour, comparing actual user behaviour to those profiles, and flagging deviations from the normal (Bace, 2000). The aim of this approach is thus to verify the identity of each user on the system through comparing current user behaviour against historical user behaviour.

For this approach to be successful, one has to investigate the total behaviour pattern of users when interacting with a computer system. The total behaviour pattern of user interactions consists of two parts, that is, the behaviour of the user and the behaviour of the computer system. Up to now, the behaviour of the user was mostly used to detect abnormal behaviour, for example, the set of typical commands being used by the particular user and the frequencies with which they are being utilized, were used to identify the abnormal behaviour. The behaviour of the system can be defined in terms of the system responses to the user behaviour. For example, if the user is allowed to use an MS Word program, the memory usage and the processor power for the Word program can represent the behaviour of the computer system.

Furthermore, the total behaviour pattern of users must also include the needs of every user on the computer system. For example, some users will mostly use the system to send and receive e-mails while other users in the same organisation, such as secretaries, might use it to type and edit documents. The needs of a user should be directly proportional to the time spent by the user on the system, thus for secretaries, for example, the time elapsed for the Word program should represent the needs of the secretary for the system.

Various methods, such as statistical techniques, were investigated to implement this approach. The main disadvantage associated with this technique is that one must define certain statistical assumptions that might influence the accurateness of the engine. With neural networks, such assumptions are not needed and therefore it was decided to implement this approach through an artificial neural network. A neural network is a set of simple elements (neurons) each linked to some of the others, that transmit information to each other through links (connections). The network consists of various input and output neurons and uses certain weight structures and thresholds to predict an output (Murray, 2000).

The neural engine implements the multi-layer perceptron (MLP) network and is implemented through two stages.

X In the first stage, the engine is populated by a training set of historical sample data that is representative of the total user behavior. This training process is only performed once for each user on the system and thereafter the historical behavior pattern (reference) is updated periodically. The supervised learning strategy is used to populate the engine and the mathematical chain rule for differentiation is used for the learning part (Murray, 2000). The mathematical chain rule can be mathematically represented by the following expressions:
For a set of N patterns of a training session the mean square output error is

$$\varepsilon = \frac{1}{2N} \sum_{\text{patterns } p} \sum_{\text{outputs } k} (V<_{kp} - V_{kp})^2 \quad \ldots \quad (1)$$

where N = the number of patterns
$V<_{kp}$ = the desired output
V_{kp} = the actual output
and the error is made smaller by the following expression

$$\Delta T_{ab} = - \eta \frac{\partial \varepsilon}{\partial T_{ab}} \quad \ldots \quad (2)$$

where T_{ab} = the applied weight.

X In the second stage, the engine accepts input data (event data) and compares it to historical behavior references, determining similarities and differences. This process will be conducted on a continuous basis.

The output of this behaviour recognition process is a numeric value: 1 if the user behaviour has changed over a period of time and 0 if the user behaviour did not alter over a period of time.

To summarise the neural approach, the total behaviour pattern of user interactions, which consists of 1) the user behaviour 2) the computer system behaviour and 3) the user needs, are used to determine whether a user is the "correct - legitimate" user. The approach is implemented by the neural engine through two basic steps. During the first step a reference behaviour pattern is built for each user on the system and during the second step the current behaviour of each user is compared against the reference pattern. The result of the comparison process is sent to the central analysis engine. NeGPAIM's fuzzy engine controls the update of the reference behaviour pattern, thereby ensuring that the intruder cannot train the intrusion detection system to accept abnormal behaviour as normal. The implementation of the neural engine was done with an off-the-shelf product called Q-NET. The implementation details of the neural engine will be described in the next section by means of the implementation model.

5 THE EXPERIMENT

An important step in creating any new approach is proper testing. To test the neural approach thoroughly, it was important to implement a working prototype in a typical real-life environment. The objective of the experiment was two-fold, firstly to determine whether the neural network can be trained and used to accurately predict abnormal user behaviour and secondly to minimize the false alarm rate to an acceptable level, typically less than ten percent.

The neural network program, Q-NET, has been designed to provide both the expert and the novice with a powerful tool for creating and implementing back-propagation style neural networks into everyday problem solving such as the intrusion detection problem.

The Q-NET program was implemented in a Windows 2000 client-server environment through an implementation model. The implementation model consists of four steps:

- Data Collection and Preparation;
- Creating a Neural Network Model;
- Training the Model; and
- Testing the Model.

The first step in the implementation process is collecting a set of training examples. Most of the data for the training examples was obtained from the various operating system audit logs. For the rest of the data, a program in the form of a windows service was developed and loaded on all the host computers. As soon as a user interacts with the system, the windows service is triggered and the necessary data is sent to a central database. The SQL Server 2000 Database Management System manages the data. For example, one particular user behaviour pattern could be represented as a string of values as follows:

Process	**Behaviour Element**	**Behaviour Element**	**Behaviour Element**
Process Name	Time Elapsed	Memory Used (Kb)	Etc. ….
MS Word.exe	4000	1438	…..

Table1: User Behaviour Pattern

This behaviour pattern defines typical behaviour elements when the process, MS Word.exe is used. From this string, the refiner must create the training examples. The refining process includes the transformation of the string values from nominal values to numeric values. For this prototype, the twelfth most commonly used processes were selected and together with the other behaviour elements (such as time elapsed and processor power), an eighteen byte input string, also referred to as the input vector, was constructed. Instead of using the actual name of the process for the input vector, the process can be represented by a series of 0's and a single "1". When analysing table 2, inputs 1 to 12 will represent the twelve processes. For this particular input, “1,0,0,0,0,0,0,0,0,0,0,0”, will represent the MS Word.exe process.

For the continuous data, such as the time elapsed, the total time spectrum was divided into eleven non-linear intervals. For the time elapsed example, a time interval between 1 and 10 seconds will represent a value of 0.1 and for a time interval between 11 seconds to 50 seconds, a value of 0.2 is allocated. Thus, if a user executes the Word process for 46 seconds, then a value of 0.2 will be allocated.

An example of a particular user behaviour input vector could be represented as follows:

Process												**Behaviour Elements**					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0	0	0	0	0	0	0	0	0	0	0.2	0.3	0	0	0	0

Table 2: Input Vector

The final part of the data preparation process is to tell the neural network what the output should be for every given input vector. The output will be between 0 and 1. For example, if the input vector corresponds exactly to a legitimate user behaviour pattern, then the output should be equal to “1”. If the input vector is completely different to a legitimate user behaviour pattern, then the output should be equal to “0”.

The final input-output vector for a particular user behaviour pattern could be as follows:

Input	**Output**
“1,0,0,0,0,0,0,0,0,0,0,0,0.2,0.3,0,0,0,0”	“1”

Table 3: Input-Output Vector

The second step is to create the neural network model. The standard five-layer back-propagation architecture was chosen for the neural network. The idea was to get results on the most standard and general architecture so that the feasibility of the approach could be demonstrated and the results could easily be replicated. More sophisticated architectures could be used and they would probably lead to slightly better results. The input layer consists of 18 units, representing the user behaviour pattern; the hidden layers have 10 units and the output layer has 1 unit. Figure 2 models this network.

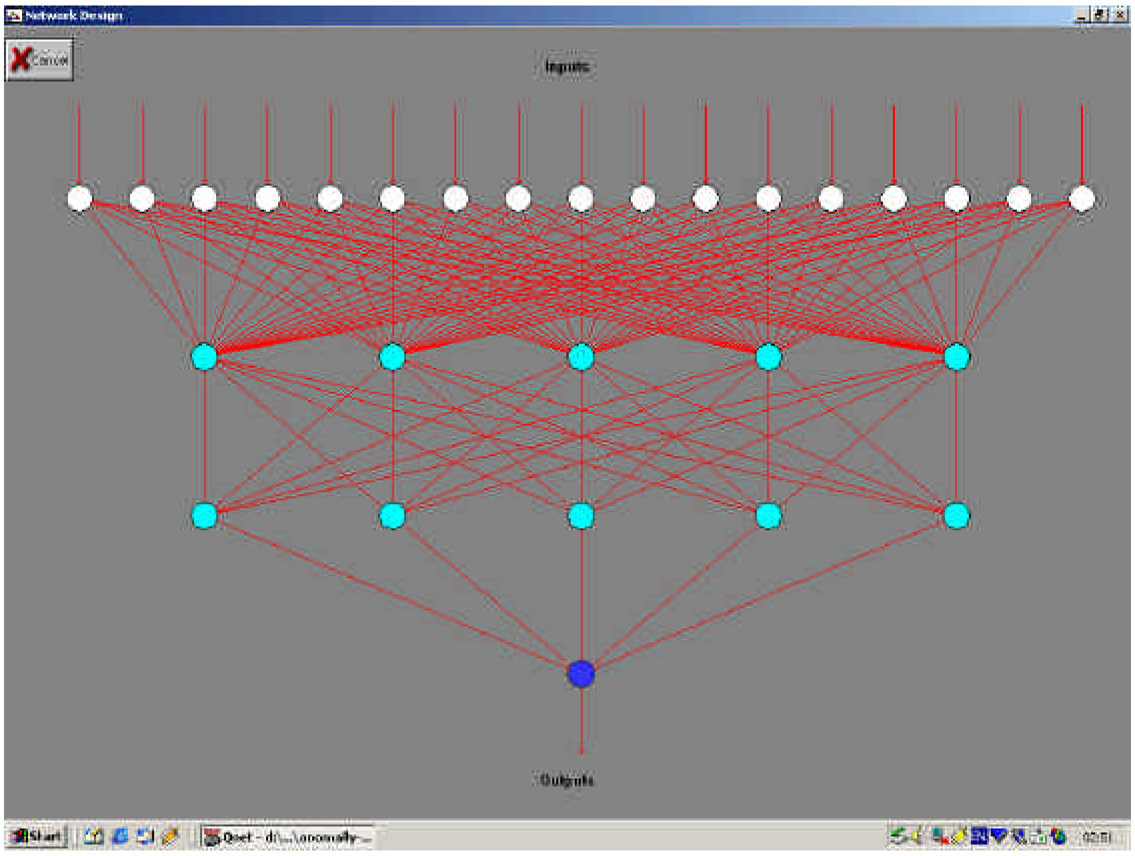
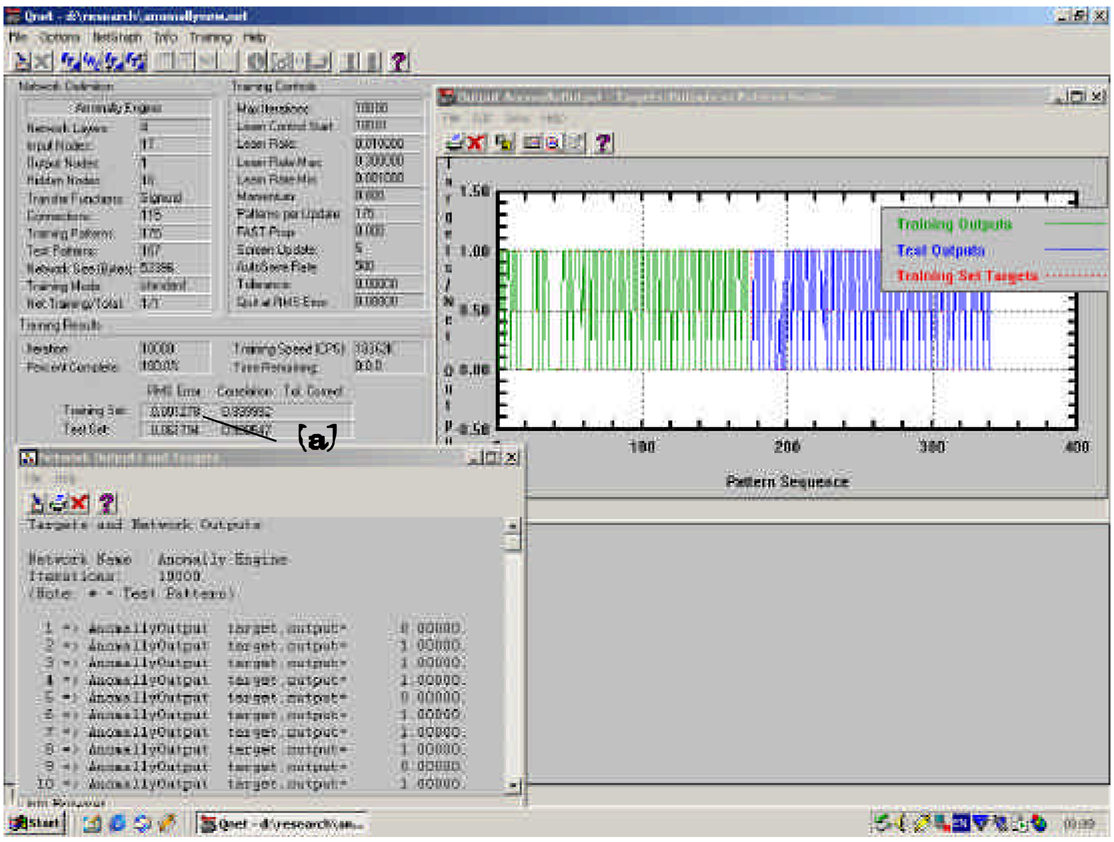


Figure 2: Neural Network Model

Training the neural network is the third step. With the training examples created as mentioned during the explanation on the first step, the next step was to extract a training set, test set and production set. The training set was used to actually train the neural network, while the test set was used to verify that the neural network function correctly. The training process continued until the lowest average level of error is reached. The weights that produced the lowest average error on the test set are kept. The final architecture and weights represent the trained neural network model. The production set was used at the end as a final test of how accurately the model actually identifies the abnormal user behaviour. Figure 3 shows the results for the training process. It can be seen from the graph, see pointer (a), that the model has a 0.999992 correlation between the target result and the model result, which indicates that the model is properly trained.



Network Definition

Anomally Engine	
Network Layers:	3
Input Nodes:	17
Output Nodes:	1
Hidden Nodes:	16
Transfer Functions:	Sigmoid
Connections:	115
Training Patterns:	175
Test Patterns:	167
Network Size (Bytes):	52396
Training Mode:	standard
Net Training/Total:	1/1

Training Controls

Max Iterations:	10000
Learn Control Start:	10001
Learn Rate:	0.010000
Learn Rate Max:	0.300000
Learn Rate Min:	0.001000
Momentum:	0.800
Patterns per Update:	175
FAST Prop:	0.000
Screen Update:	5
AutoSave Rate:	500
Tolerance:	0.00000
Quit at RMS Error:	0.00000

Training Results

Iteration	10000	Training Speed (CPS)	333628
Percent Complete	100.0%	Time Remaining	0:0:0

	RMS Error	Correlation	Tol. Correct
Training Set:	0.001278	0.999992	
Test Set:	0.062194	0.980547	

Network Outputs and Targets

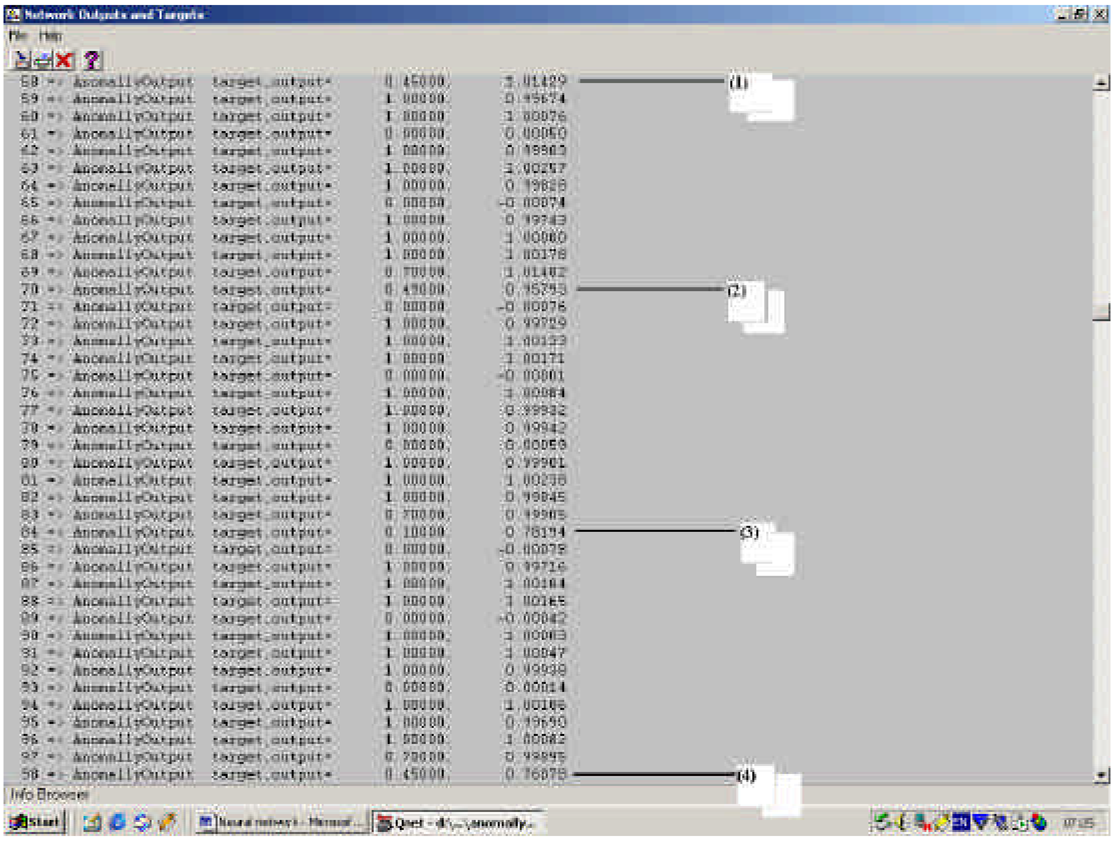
```
Targets and Network Outputs

Network Name   Anomally Engine
Iterations     10000
(Note: * = Test Pattern)

  1 => AnomallyOutput   target,output=     0.00000,
  2 => AnomallyOutput   target,output=     1.00000,
  3 => AnomallyOutput   target,output=     1.00000,
  4 => AnomallyOutput   target,output=     1.00000,
  5 => AnomallyOutput   target,output=     0.00000,
  6 => AnomallyOutput   target,output=     1.00000,
  7 => AnomallyOutput   target,output=     1.00000,
  8 => AnomallyOutput   target,output=     1.00000,
  9 => AnomallyOutput   target,output=     0.00000,
 10 => AnomallyOutput   target,output=     1.00000,
```

Figure 3: The training process results

The final step is to test the neural network with the production test set. Figures 4 and 5 show the results of this test.



```
58 => AnomallyOutput  target,output=  0.45000,  1.01429
59 => AnomallyOutput  target,output=  1.00000,  0.99674
60 => AnomallyOutput  target,output=  1.00000,  1.00076
61 => AnomallyOutput  target,output=  0.00000,  0.00050
62 => AnomallyOutput  target,output=  1.00000,  0.99963
63 => AnomallyOutput  target,output=  1.00000,  1.00257
64 => AnomallyOutput  target,output=  1.00000,  0.99828
65 => AnomallyOutput  target,output=  0.00000, -0.00074
66 => AnomallyOutput  target,output=  1.00000,  0.99743
67 => AnomallyOutput  target,output=  1.00000,  1.00080
68 => AnomallyOutput  target,output=  1.00000,  1.00178
69 => AnomallyOutput  target,output=  0.70000,  1.01402
70 => AnomallyOutput  target,output=  0.45000,  0.95793
71 => AnomallyOutput  target,output=  0.00000, -0.00076
72 => AnomallyOutput  target,output=  1.00000,  0.99729
73 => AnomallyOutput  target,output=  1.00000,  1.00133
74 => AnomallyOutput  target,output=  1.00000,  1.00171
75 => AnomallyOutput  target,output=  0.00000, -0.00001
76 => AnomallyOutput  target,output=  1.00000,  1.00084
77 => AnomallyOutput  target,output=  1.00000,  0.99932
78 => AnomallyOutput  target,output=  1.00000,  0.99942
79 => AnomallyOutput  target,output=  0.00000,  0.00059
80 => AnomallyOutput  target,output=  1.00000,  0.99981
81 => AnomallyOutput  target,output=  1.00000,  1.00238
82 => AnomallyOutput  target,output=  1.00000,  0.99845
83 => AnomallyOutput  target,output=  0.70000,  0.99985
84 => AnomallyOutput  target,output=  0.10000,  0.78134
85 => AnomallyOutput  target,output=  0.00000, -0.00079
86 => AnomallyOutput  target,output=  1.00000,  0.99716
87 => AnomallyOutput  target,output=  1.00000,  1.00184
88 => AnomallyOutput  target,output=  1.00000,  1.00165
89 => AnomallyOutput  target,output=  0.00000, -0.00042
90 => AnomallyOutput  target,output=  1.00000,  1.00083
91 => AnomallyOutput  target,output=  1.00000,  1.00047
92 => AnomallyOutput  target,output=  1.00000,  0.99938
93 => AnomallyOutput  target,output=  0.00000,  0.00014
94 => AnomallyOutput  target,output=  1.00000,  1.00186
95 => AnomallyOutput  target,output=  1.00000,  0.99690
96 => AnomallyOutput  target,output=  1.50000,  1.00082
97 => AnomallyOutput  target,output=  0.70000,  0.99895
98 => AnomallyOutput  target,output=  0.45000,  0.76078
```

Figure 4: The Test Results

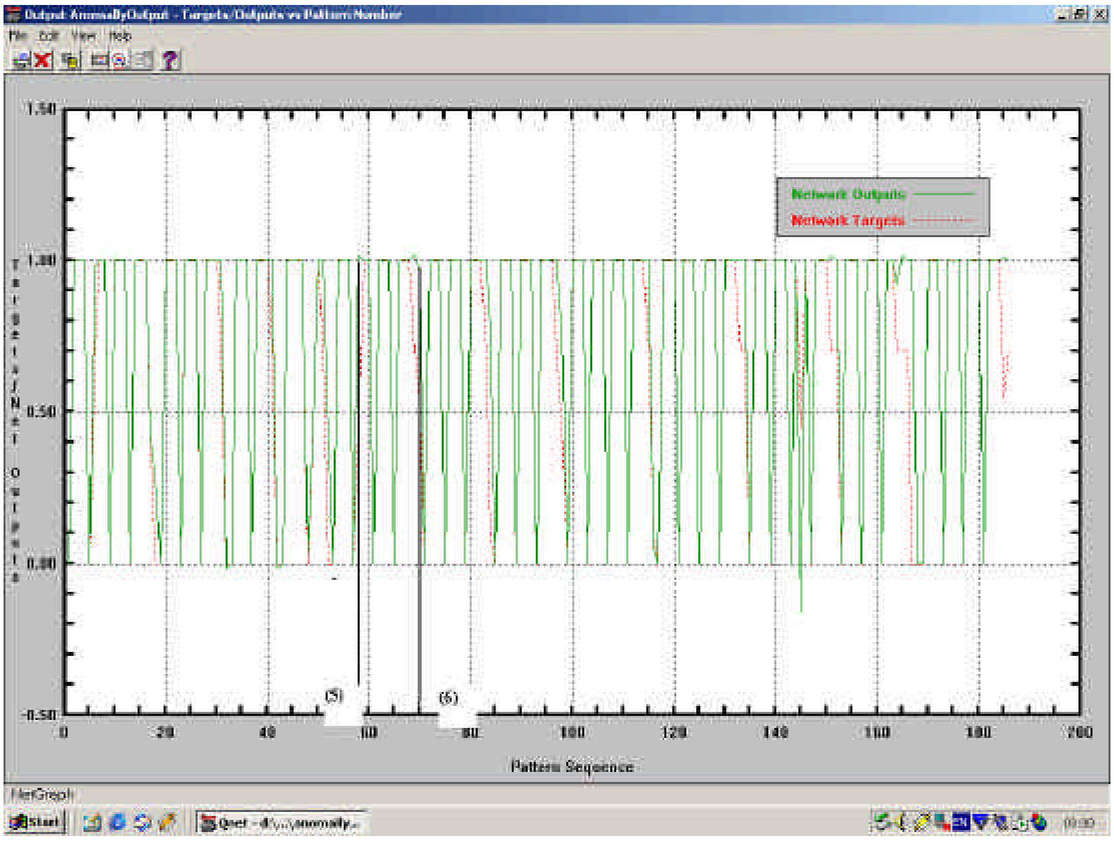


Figure 5: The Target/Output Test Graph

Examining the production set testing results would seem to indicate that the neural network, which implements the neural approach, performs relatively well. The neural network identifies 45 abnormal user behaviour patterns out of a possible 186 patterns. The network was more than 97% accurate in detecting unusual activities, with less than 5% false alarm rate. Figure 4 and 5 show the correlation between the predicted network outputs and the true target outputs. According to figure 4, all the patterns' network outputs, except patterns 58, 70, 84 and 98, are in correlation with the true target outputs (see pointers 1-4). Figure 5 provides a graphic representation of the results and confirm that pattern 58, see pointer 5, has a target output of 0.45 but the network is equal to 1.01, also pattern 70, see pointer 6, has a target output of 0.49 but the network output is equal to 0.95. Thus if one examines these two patterns, one can conclude by saying that both the behaviour patterns were non-legitimate user behaviour patterns, but the neural network incorrectly predicted that they were legitimate behaviour patterns.

While these results appear favourable and can be utilized in an intrusion detection system, the next step will be to automate and streamline the learning process of the neural network, which is controlled by the fuzzy engine, so that it can be tested on a bigger scale and in a true-life environment. To conclude, although this experiment was done on a very small scale, the results seem significant and prove that the neural approach as defined in section 4 has the potential to overcome one of the shortcomings of current anomaly intrusion detection system; that is, complex and high computational overheads and incorrect training process.

6 THE CONCLUSION

With the increasing number of breaches in security, intrusion detection has become very important for a large number of organizations. There are many different aspects of intrusion detection to be considered. Most commercial systems utilize a misuse analysis engine. They can therefore only detect known intrusion attacks. Anomaly detection systems are unreliable and have high overheads and few commercial solutions implement them. To increase the reliability of intrusion detection systems, a new approach is required. This approach should be able to obtain information from a wide variety of sources, and employ both analysis and misuse detection to form a hybrid system. An alternative model, called NeGPAIM, has been proposed to overcome these shortcomings.

The neural engine was highlighted as one of the three main components of NeGPAIM. This component implements the neural approach as outlined in section four. The approach uses the total behaviour pattern of user interactions to determine whether a user is the "correct - legitimate" user. The approach consists of two steps. During the first step a reference behaviour pattern is built up for each user on the system and during the second step the current behaviour of each user is compared against the reference pattern.

The neural approach was implemented through an initial working prototype. Although the prototype was done on a small scale, favourable results were obtained. The network was more than 97% accurate in detecting unusual activities, with less than a 5% false alarm rate.

To conclude, more tests and research are currently being conducted on NeGPAIM and the prototype. The ultimate objective of this project is to streamline this model through practical implementations and to utilize this model to combat intrusion attacks proactively.

7 REFERENCES

Anderson, J.P. (1980). Computer Security Threat Monitoring and Surveillance. [Electronic Version]. James P. Anderson Co.

Bace, R.G. (2000). Intrusion Detection. Macmillan Technical Publishing, Indianapolis, In USA.

Bruneau, G. (2001). The History and Evolution of Intrusion Detection. Sans Institute. Retrieved 20 May, 2002 from http://rr.sans.org/intrusion/evolution.php

Botha, M & von Solms, R. (2001). Utilization of Trend Analysis in the Effective Monitoring of Information Security: The Concept (Part One). Information Management and Computer Security.

Botha, M & von Solms, R. (2002). Utilization of Trend Analysis in the Effective Monitoring of Information Security: The Model (Part Two). Information Management and Computer Security.

Fred Cohen & Associates. (1996). Intrusion Detection and Response. Fred Cohen & Associates. Retrieved 20 May from http://all.net/journal/ntb/ids.html

Mc Hugh, J. (2001). Intrusion and Intrusion Detection. International Journal of Information Security.

Murray, A. (1996). Applications of neural networks (1st ed.). Netherlands: Kluwer Academic Publishers

Pastore, M. (1999). "Higher Growth Rates Predicted for B2B E-Commence", from http://idm.internet.com/articles/199912/na_12_29_a.html

Peppard, J. (1993). Information, technology and strategy. In J. Peppard (Ed.), I.T. strategy for business (pp1-25). London : Pitman Publishing. [ISBN: 0-273-60024-9].

Power, R. (2001). 2001 CSI/FBI Computer Crime and Security Survey. Computer Security Issues and Trends, Vol VII, No 1.

Price, K. (1996). Intrusion Detection. Coast. Retrieved 20 May 2002, from http://www.cerias.purdue.edu/coast/intrusion-detection/detection.html

Richardson, R. (2002). "Computer Crime Bleeds U.S. Corporations, Survey Shows; Financial losses from attacks." [on-line]. Available from the Internet: URL

Http://www.gocsi.com/press/20020407.html.

Seleznyov, A. (2000). "A Hybrid Model for Intrusion Detection Based on Temporal Probabilistic Networks", IFIP `2000.

Seleznyov, A. (2002). "Anomaly Intrusion Detection System Based on Online User Recognition", INC `2002.

Smaha, S.E. (1988). Haystack: An Intrusion Detection System. Tracor Applied Science Inc., Austin, Texas. In Fourth Aerospace Computer Security Applications Conference, pp. 37-44.

Sundaram, A. (1996). An Introduction to Intrusion Detection. ACM. Retrieved 20 May 2002 from http://www.acm.org/crossroads/xrds2-4/intrus.html

Von Solms, R. (1993). "A process approach to information security management" , IFIP'93, WG 11.1, Toronto, Canada, 1993.

8 PERMISSIONS

We would like to thank the National Research Foundation for their financial support for research conducted towards this paper.