# NETWORK-BASED INTRUSION DETECTION USING NEURAL NETWORKS

**CHANDRIKA PALAGIRI**
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
palagc@cs.rpi.edu

***ABSTRACT***
With the growth of computer networking, electronic commerce, and web services, security of networking systems has become very important. Many companies now rely on web services as a major source of revenue. Computer hacking poses significant problems to these companies, as distributed attacks can render their cyber-storefront inoperable for long periods of time. This happens so often, that an entire area of research, called Intrusion Detection, has been devoted to detecting this activity. We show that evidence of many of these attacks can be found in a careful analysis of network data. We also illustrate that the learning abilities of neural networks can serve to detect this activity. We test our systems against denial of service attacks, distributed denial of service attacks, portscans, and even some doorknobs attacks. Finally, we also show how our systems detect long-term attacks, which occur when attackers space out their efforts to evade detection. In this work, we explore network based intrusion detection using a Perceptron-based, feed-forward neural network system and a system based on classifying, self-organizing maps. Both of these systems are tested on data provided from the DARPA intrusion detection evaluation program as well as live attacks in an isolated computer network.

## INTRODUCTION

Intrusion Detection is a relatively new field which tries to detect computer attacks from examining various data records observed by processes on the same network. These attacks are normally split into two categories, host-based attacks and network-based attacks (Branch, 2002). Host-based attacks are generally attacks that target a machine on a network. These attacks are used to gain access to some feature of the machine under attack, such as user accounts or files on the machine. Network-based attacks are types of attacks that generally cause some denial of service to a machine on a network, by disallowing normal or legitimate users access to services on a machine, or by slowing down the netowork connectivity on a network, so that legitimate users cannot do what is needed as the network is being attacked. Host-based detection routines normally use system call data from an audit-process that tracks all system calls made on behalf of each user on a particular machine. These audit processes must be run on each monitored machine. Network-based attacks

detection routines typically use network traffic data from a network packet sniffer. Many organizational computer network protocols including the widely accepted Ethernet (IEEE 802.3) protocol use a shared medium for communication. Therefore, the packet sniffer only needs to be running on the same shared subnet as the monitored machines. In addition, we plan to use a tool developed within our group, called DOORS (Bivens, 1999a; Bivens, 1999b; Bivens, 2002) for collecting the network traffic data at many routers. Using the process of sniffing packets on a network, we believe that enough data can be gathered in order to build a Perceptron-based feed-forward neural network system to detect the occurances of attacks against a network. The premise behind creating such a system is that most network based attacks usually flood a network with a large number of packets, which are higher than the packets seen when an attack is not occurring. This system will not prevent the attacks to a network, which is very hard to do, but it will alert the administrator that there is suspicious/anolomous activity occurring on the network.

**ATTACKS THE SYSTEM SHOULD DETECT**

As stated above, the neural network will be used for the detection of denial of service attacks. Some attacks and their description to be detected by the neural network are as follows:

- **SYNFLOOD** – This attack affects every operating system that implements the TCP protocol. This attack bombards a machine by sending dozens of falsified connection requests in short periods of time to prevent legitimate connection requests. These connections can be distinguished by the number of times the service is requested by the attacker.
- **UDPSTORM** – This attack uses the UDP protocol to create denial of service. The attack is usually an echo-echo attack that creates communication between two echo ports on two different machines causing a great number of packets to be sent from machine to machine since the echo ports on both machines will keep echoing each other. With this, slowing of the network is ensured and this prevents normal users from accessing machines on a network upon which the attack is unleashed.
- **SMURF** – This attack uses ICMP broadcast addresses to unleash its power on a victim machine. The attack sends ICMP echo requests to many IP broadcast addresses. Every machine that is listening and is on the network that contains the targeted broadcast address will respond by sending ICMP echo replies back to the victim. This is a very large attack for the reason that there can be as many as 255 hosts on a subnet that can respond to the echo requests, and if there are many broadcast addresses used in the attack, the replies can increase by a magnitude of 255 per broadcast address, causing a great number of packets destined to the victim, which in turn causes the victim to be flooded, possibly denying services to legitimate users.

**BACKGROUND**

There are a few groups that have attempted and successfully used neural networks for intrusion detection, each promoting a different approach. At MIT Lincoln Laboratory, a neural network was applied to misuse detection. The data they obtained for the neural network consisted of attack-specific keywords in network traffic from Unix-host attacks and attacks that allows the attacker to obtain root privileges on a server (Lippmann, 1999). In order to see the attack keywords in the network traffic, the payload of the packets needs to be evaluated. The differences between this neural network and the neural network for the project are not only the fact that they don't look at the hits to the ports, like we do, but also, they look into the payload of the network traffic in order to pick out the keywords that can be used in an attack. The neural network detected 17 out of 20 attacks and gave one false alarm. At UBILAB Laboratory, a Self-Organizing Map approach was used along with a neural network. The self-organizing map clustered the network traffic in to two-dimensional spaces for visualization (Girardin, 1998). This process is similar to our project except that they use the visual effects of self-organizing maps, whereas for this project, we want the self-organizing map to tell us to which cluster the data belongs. Also, they would need someone to look at the graphical representation of the data and have that person determine if an attack is occurring. This project would not require such human involvement. The result of this approach is that it detected IP spoofing, FTP password guessing, network scanning. In a different approach from misuse, Reliable Software Technologies created a neural network for anomaly detection. It analyzed program behavior profiles, which are system calls made by programs running on a system (Ghosh, 99). Another visualization tool used for intrusion detection is a clustering of network traffic (Oliver, 2001). This is an example of the use of a tool to cluster network traffic together. This tool, by aggregating clusters of network traffic, can help an administrator to detect network anomalies. The difference between this method and the method described in this paper is that they require a visualization of the traffic for the administrator to notice an anomaly, but this project doesn't require that, since it will incorporate a neural network to notice the anomaly.

**DEFINITION OF NEURAL NETWORKS**

Neural networks also known as artificial neural networks are used to allow computers to learn and adapt to different tasks that they are presented with. It can be seen as a computer representation of the human brain. In the human brain, there are neurons interconnected to each other, and depending on an impulse the neurons receive, a response will occur. This impulse that neurons in the human brain receives is equivalent to the input that is given to an artificial neural network, so as the response that follows an impulse is similar to a decision the neural network outputs.

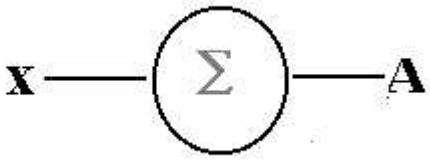


**Figure 1 – A single neuron for a neural network taking input x with an output A**

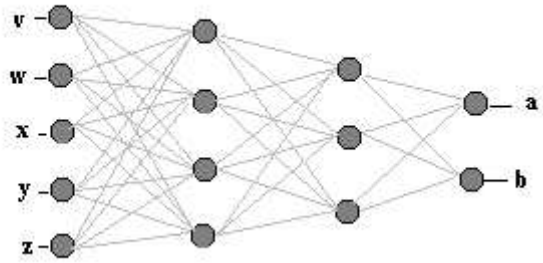


**Figure 2 – Graphical representation of neurons interconnected receiving many inputs and producing two outputs**

There are many different types of neural networks, and discussing them is beyond the scope of this paper. Instead, we focus on two types of neural networks that are used for this project: the multilayer feed forward perceptron and the self-organizing map.

The multilayer feed forward perceptron neural network can be seen as a fully connected graph. Similar to the neurons in the human brain, the neurons in a neural network take the input that is given to them and multiply each input by a weight factor, take the sum of the results of the multiplication and send the result forward to the next set of neurons. To make decisions, the neural network, again similar to the human brain, needs to learn as much as it can about the subject or problem it needs to solve. It is similar to a young child learning his or her ABCs'. Sample data with end results must be given to the neural network before it can make decisions. The way this learning period occurs is by using the process of traversing the sum of the weights of the inputs, throughout the network until the resulting factor is close enough to the final result (within some prescribed error). To get the final result, each node in the network periodically adjusts the weights that are multiplied to the input, resulting in values as close to the final result that was told to the neural network in the beginning stages as

possible. After learning, the neural network can be given problems that are similar to the ones that it was trained on and it would make decisions about the data that it is currently processing. This whole process of training and manipulating a neural network can be said as: the neural network is in the beginning to be presented with as much data as you can about the subject that you want it to learn and when it encounters a similar problem that it has never encountered before, it will make a decision on what the solution should be (fortunecity.com). The neural network of this kind was successfully used for detection of congestion sources in computer networks (Bivens, 2000a, Bivens, 2000b, Bivens, 2002).

The self-organizing map is a neural network that organizes data into groups that contain similar attributes. It takes patterns of arbitrary dimensions and transforms them in a one or two-dimensional map (Haykin, 1999). It contains neurons just as the multilayer feed forward perceptron neural network, except they make decisions differently. The difference is that the self-organizing map is a self-learning neural network, unlike the multiplayer feed forward perceptron. It is self-learning by not requiring an example of the output in order to make decisions, but requires the neurons to compete with each other about who represents the data better and the winner is seen as the neuron that matches the data the best. This way, the data will be formed into clusters of data that are somehow related to each other. Another difference between a multiplayer feed forward perceptron and a self-organizing map is that a multilayer feed forward perceptron is static being inflexible in the number of inputs it can receive, while the self-organizing map is not due to the self-learning and clustering structure of the map.

**METHODOLOGY**

The most basic anomaly intrusion detection system consists of a rule-based system, where there are rules that define signatures of known attacks. Whenever network packets enter the networks that use the rule-based system, they evaluate the packets and apply the rules of the known attacks to the packets to see if there are any matches. If there is a match, an alert is made. A problem with this system, is that if an attack that has no rules in the database is attacking the network, no alerts will be made until after the attack has finished running the course and the rules of the attack are added to the attack database. This type of system, cannot detect new attacks, only attacks that it has seen before. The use of neural networks for intrusion detection was chosen because intrusion detection is a complex problem, and neural network are used to solve complex problems. Also, the power of neural networks to make sound decisions about the problem it is knowledgeable about made them a great candidate for detecting anomalous patterns in a network. The multilayer feed forward perceptron was one of the neural networks chosen for this project because it is the neural network that is the participants on this project are most knowledgeable about. Professor Mark Embrechts, a professor in the Decision Sciences and Engineering Sciences department, developed the neural network software used

for this project. Having the access to the software allowed us to use a tool that supplies sufficient results rather than having to create our own tool. The second neural network that was chosen is the self-organizing map. It was decided that the data that will be supplied to the multilayer feed forward perceptron would need to be correlated in some way. Since the self-organizing map creates a correlation between data by grouping the inputs that are similar to the same class/cluster together, it seemed to be the best tool for correlating the data that will be given to the neural network.

**BUILDING NEURAL NETWORK STRUCTURE**

Prior to collecting and monitoring the network traffic which we believe holds the information about intrusions, we must first determine the structure of the neural network. We present the current trend in network traffic to the neural network in the form of the number of times a host is accessed across the network in a certain time interval $\Delta t$. To allow a machine to differentiate one application's traffic from another, hosts have many ports responsible for services such as telnet and ftp, to which packets must travel to and from. Most networking protocols use a 16-bit port number which facilitates a possible $2^{16}$ or 65,536 ports that receive network traffic at any given time (Stevens, 94). Monitoring all of these ports through a nueral network is not only infeasable, but also unneccesary. It is highly unlikely that all ports on a particular host are used, since the services available are turned off by the person(s) in control of the machines on a network. Therefore, we must establish which ports are important for us to monitor.

To determine the important ports to monitor and thus determine the initial architecture of our neural network, we have a architectural learning phase before our normal neural network learning phase. We first establish an architectural multiplier $F$ wich is multiplied by the time interval $\Delta t$ to develop the length of our architectural learning phase. The network traffic is then observed for $F*\Delta t$ time to catalogue the number of times source hosts access the many different ports on the target machine is. The ports accessed in this period of time form the set $A$. In the beginning, the administrator gives the architectural learning phase a list of known ports to watch (the set $KP$), the number of extra ports the algorithm can choose to add to the $KP$ list $ep$, and the number of source clusters the system should develop $sc$. At the end of our architectural learning phase, the known ports given in the beginning are weeded out from the accessed ports into a set of remaining ports ($REMAINING = A - KP$) and the top $ep$ ports are taken from the set $REMAINING$ to complete the set of ports that will be monitored by the system. It is called the finalset ($FINALSET = KP \ 4 \ max(ep, A - (KP \ 3 \ A))$) where the function max($x$, $Set\ x$) returns a set containing the $x$ highest elements of $Set\ S$.

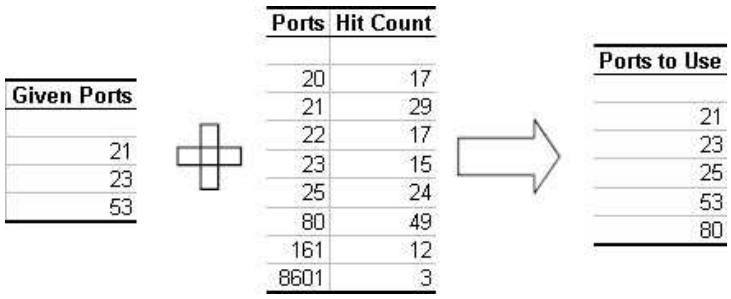


**Figure 3 – Process of obtaining *FINALSET***

The above figure shows that the administrator first chose ports 21, 22, and 23 for set *KP*, at the end of $F^*\Delta t$, set A contains the ports and the number of hits the machine received during $F^*\Delta t$. The administrator also told the architectural algorithm to add 2 extra ports. With that information, the final set *Final Set* now contains *KP* and extra (2) highest elements of A which are ports 25 and 80.

Any machine on a network usually receives some type of connections from other hosts that are either from the same subnet or different. Hence, we decided to gather port information on a per source basis during the beginning of the neural network learning phase of the project. This phase gathers the total hits per source to the monitored machine during interval $\Delta t$. The total hits to the ports in *FinalSet*, per source, creates an input size of $N * M$ for the neural network, where $N$ is the number of ports in *FinalSet* and $M$ is the number of sources seen in $\Delta t$. The first M nodes of the neural network's input receive the totals of the hits to the ports in *FinalSet*, from the first source. The next M nodes will receive the respective totals for the second source in the same order as the first. An example of this architecture where there are 4 sources ($N$=4) and 5 ports ($M$=5) from the *FinalSet* from the previous figure can be found in the following figure.

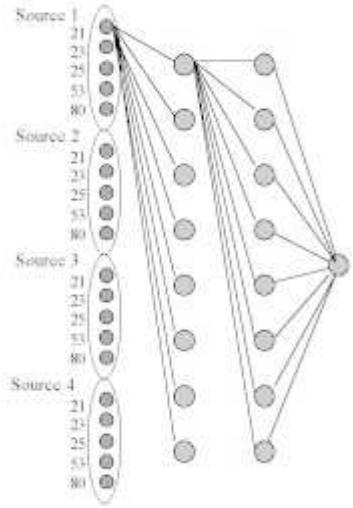


**Figure 4 – Neural Network with 4 sources**

Gathering total hits per source during $\Delta t$ creates a problem for the neural network. The number of sources received may change frequently from one interval to another interval, creating different sizes in the number of inputs the neural network would receive. For example with a *FinalSet* containing 4 ports:

> At the end of the first $\Delta t$, 3 sources made connections to destination x, creating an input size of 12 for the neural network. At another $\Delta t$, 6 sources made connections to destination x, creating an input size of 24 for the neural network.

This is a huge problem for the multilayer feed forward perceptron due to the fact that the neural network is created on a static set of inputs, making it inflexible to the number of input it receives. This inconsistency at any given $\Delta t$, of the number of sources seen, renders the neural network useless. At any given point, the observed source quantity could rise far above or sink far below the N input nodes used by the neural network. To allow for this incompatibility, the N sources are expanded to represent N clusters or N classes of sources. Source traffic data, which is the hits to the monitored ports, is aggregated into the appropriate clusters and the aggregate cluster statistics are used as inputs to the neural network.

The technique used to cluster the source traffic data, is done by a self-organizing map, which selects the cluster to which the source traffic data belongs. Only data gathered in the given $\Delta t$ is clustered together, if possible, and is placed as input for the neural network. For example, if there are only 3 inputs to the neural network, and there are 4 sources in one $\Delta t$, the self-organizing map would place each source's data into one of the 3 clusters and then aggregate the data in the clusters. This aggregation of the data will then be

sent to the neural network for analysis. This clustering approach would guarantee that no matter what the number of sources obtained per $\Delta t$ is, the number of inputs to the neural network would remain fixed to N * M. After collecting all of the information and clustering by traffic from the sources, the neural network is then trained on the data, in order to make decisions during monitoring, of whether the data currently being looked at is an attack or not.

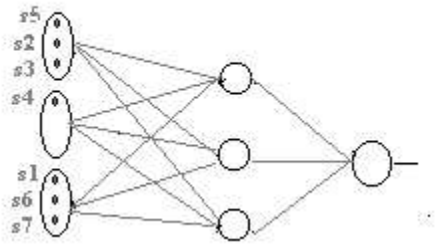


**Figure 5 – Example of clusters of sources as input for the neural network**

Figure 5 is an example of a graphical representation of the neural network. It contains 3 clusters of aggregated sources and their hits to the monitored ports, as input. It also has one hidden layer and one output layer, which outputs the decision whether the data seen has attributes of an attack or represents normal traffic to the host. The self-organizing map is the tool that decides in which cluster to put the source hit information. That is the reason why source 1 is in cluster 3 and source 5 is in cluster 1.

## CONTRIBUTIONS MADE TO THE PROJECT

### The Java Expert

We are using the feed forward neural network with back propagation, "Meta Neural", created in C, by Dr. Mark Embrecht, for this project. This is the neural net that we train on the network data. To make the program compatible with existing intrusion detection agents, however, we have a Java expert that can take the weight file created by the neural network after the training, and use that for the testing phase. That is to say, Meta Neural builds the structure of the network, after seeing a number of examples for normal and attack data, and this structure (weight file) is used by the expert to apply various rules on the network traffic, and determine whether or not it may correspond to an attack. The Neural network is created according to the specifications (the number of inputs and outputs) and trained using Meta Neural. The trained Neural Net is saved on disk as a file. The Java expert uses this file to compute the output produced by the neural network, for a given input. The expert can not be directly trained to detect new types of attacks. If, after being in use for a while, the expert does not seem to detect intrusions satisfactorily, we can just retrain Meta Neural and use the expert with the new weight file.

The following classes make up the Java Neural Network Expert:

neuron.java - This class defines a single neuron in the network. Each neuron has a number of input paths, and one output path. The input path weights and the input values can be set, and the output is calculated using these values, along with the sigmoid function. The output at neuron $N_{ij}$ (i th neuron of layer j) is :

$$Sum_{ij} = I_{1j-1} . V_{1j-1} + I_{2j-1} . V_{2j-1} + \ldots + I_{nj-1} . V_{nj-1}$$

$$O_{jj} = 1 / (1 + e \wedge (- Sum_{ij)} );$$ ( Sigmoid function )

layer.java - This defines a layer in the neural network. Each layer consists of an array of neurons. The output from the neurons of one layer is the input to the neurons of the successive layer.

NeuralNet.java - A NeuralNet object contains a number of layers. It can initialize the layers, the connecting weights, and the inputs, and then calculate the output at each layer. The output at the last layer (output layer) is the final output from the neural net.

nnWeights.java - This is a serializable class which contains the structure of the neural network, stored as two arrays: the first one containing the number of neurons in each layer, and the second one containing the weights of the paths between adjacent layers. This makes it possible to save the structure of the neural network once it is trained, and to use it to calculate the output for any given input.

MyNet.java - Reads the weights from the weight file (into the nnWeights object), and uses these to initialize the NeuralNet object. This object is then used to compute the output from the neural net, for a given input vector.

**The Neural Network Trainer**

This program (nnTrainer.java) is used during the training phase of the project. It calls functions to parse the tcpdump data and collect the required information, passes this onto the clustering algorithm to group the sources with respect to their activity, normalizes the clustered data, and then calls the MetaNeural to train on the data. Since this is still the experimental phase and we are trying to determine what kind of a neural network would give the best results, we created a number of different networks, with varying number of layers, neurons per layer and permissible error, and train each of them on the same data, to see what structure may be most suitable. Various objects generated during this phase are serialized for use in the actual detection phase.

The program takes the following parameters at command line: configuration file, destination, known ports, number of extra ports, architectural learning interval, time interval, number of training samples, number of test samples, fraction of normal data, number of clusters, maximum hits.

**Configuration File**: This file supplies information useful in collecting the data, such as the binary files to use, the name of the attack, the start time of the attack, and the duration of the attack.

**Destination**: The IP address of the machine to be monitored for attacks.

**Known ports**: The list of port numbers, for which we want to collect the number of hits.

**Number of extra ports - n**: Here, we tell the program to collect the number of hits for **n** most frequently hit ports, other than the ones we have already specified.

**Architectural Learning Interval**: This is the time (in seconds) that we give the program, in order to monitor all the ports and decide which are the most frequently hit ones. No data is actually recorded during this time.

**Time Interval - t**: The number of hits per each of the monitored ports is collected every **t** seconds and stored in a data structure. In order to facilitate the training of the Neural Network, normal and attack data are stored in separate structures.

**Number of training samples**: This is the number of samples (both normal and attack) that we want in the training data. A sample is the data collected in one time interval. It is usually similar to the number of test samples.

**Number of test samples**: The number of samples we want in the test data.

**Fraction of normal data**: Gives the fraction of the samples (in both training and test data) that we want to be consisting of normal data. The remaining would be attack data. If there is insufficient attack data, some normal samples are duplicated.

**Number of clusters**: It is the number of groups into which we want to cluster the sources. This decides the structure of the Self Organizing Map used for clustering, as well as the neural network used for detection.

**Maximum Hits**: This is the maximum number of hits to any port on the destination, in time interval **t**. This is required in order to normalize the number of hits. This number has been found experimentally.

We first call a function that runs tcpdump and parses the output, in order to collect the information about the specified host and ports. Along with passing the data structures with this information to the trainer, the function also serializes and stores these structures, so that we may reuse them if we want to conduct the same experiment, instead of running tcpdump again. It returns normal and attack data in two different structures. From the samples returned,

we make two sets of data, one for the training of the neural network and one for testing.

Some of the input parameters are used to initialize and train the Self Organizing Map (SOM), to be used for grouping the sources. Each data sample is clustered independently, the hosts in each being grouped according to their activity. The data is then normalized to a value between 0 and 1, to be given as input to the neural network. If the number of clustered needed is **n** and the number of ports being monitored is **p**, then the number of inputs to the neural network (NN) is **n** * **p**. This also gives the number of nodes in the first layer (input layer) of the NN. Thus, if any of the input parameters (like number of ports and clusters) is changed, both the SOM and the NN have to be reconstructed and trained again.

We create a number of different networks, with varying number of layers, neurons per layer and permissible error, and train each of them on the same data, to see what structure may be most suitable. The weight file created by the chosen NN needs to be saved for the actual working phase of Intrusion Detection Expert. The trained SOM object, port numbers, destination address, NN connection weights, maximum number of hits, and time interval are serialized (saved to a file on disk) during training, and are later used by the detector.

### The Detector

The Detector is intended to run continuously, on live tcpdump data, or archived data from files, as needed. The parser reads the tcp data, extracts the desired fields, puts them in the required data structure, and passes it to the detector after every **t** seconds. The data is then clustered, normalized, and given as input to the neural network. The output from the neural net is a value between 0 and 1, which gives the probability of the given network traffic being part of an attack. A ‘0’ means that the data is normal, and a ‘1’ means that it is definitely an intrusion.

### THE CLUSTERING ALGORITHM

Alan Bivens has written the Self-Organizing clustering technique to cluster the data for the neural network. Based on the data it uses for training, the self-organizing technique, decides the number of clusters to develop as well as which cluster the data belongs to. After the number of clusters is decided, that number become static, since the number of clusters will be the number of inputs for the neural network. Then, as new data is read by the self-organizing technique, it decides to which cluster the data belongs, which is based on previous statistics obtained through creating the initial clusters. After the cluster is chosen, the aggregation of the sources, as explained in the introduction is done and, then, sent to the neural network.

## CURRENT RESULTS

As described, our system is built to run on either historical tcpdump binaries or from a real-time tcpdump process. For testing purposes, we ran our system using tcpdump binaries from the DARPA 1999 training dataset (Lippman et al, 2000). We had a great deal of trouble trying to get the neural network to detect the collection of all attacks at once. However, the neural network performed well when trained on individual attacks. The training and testing in this area was limited however, because our dataset did not contain many instances of the same attack. A representation of some of our results can be found in Table 1 where sshprocesstable is the name of a particular denial of service attack. Testing of additional types of attacks is in progress.

| | Correct Normal Predictions | False Negatives | Correct Attack Predictions | False Positives |
|---|---|---|---|---|
| Union of All Attacks | 100% | 0% | 24% | 76% |
| sshprocesstable | 100% | 0% | 100% | 0% |

**Table 1: Current Results**

## CONCLUSIONS

Many methods have been employed for intrusion detection. However, modeling networking trends for a simple representation to a neural network shows great promise, especially on an individual attack basis. Also, using SOMs as a clustering method for MLP neural networks is an efficient way of creating uniform, grouped input for detection when a dynamic number of inputs are present. Once trained, the neural network can make decisions quickly, facilitating real-time detection. Neural Networks using both supervised and unsupervised learning have many advantages in analyzing network traffic trends and will be a continuing area of our research.

**ACKNOWLEDGEMENTS**

I would like to acknowledge Alan Bivens for his work on this project and his suggestions on what should be done, for his contribution to some of the components of the paper, and for designing the self-organizing map technique that is used in correlating the source information in a way that can be manipulated by the neural network.

I would also like to thank the following people who contributed one way or the other to this project:

Rasheeda Smith for collecting and massaging the network data for training and testing the neural network.

Dr. Mark Embrechts for creating the neural network structure that the expert is based from.

Dr. Boleslaw Szymanski for being my advisor and allowing me to work on this project.

**REFERENCES**

A. Bivens, L. Gao, M. F. Hulber and B.K. Szymanski, 1999a, ``Agent-Based Network Monitoring," *Proc. Autonomous Agents99Conference,* Seattle, WA, pp. 41-53.

A. Bivens, P. Fry, L. Gao, M.F. Hulber, Q. Zhang and B.K. Szymanski, 1999b, ``Distributed Object-Oriented Repositories for Network Management," *Proc. 13th Int. Conference on System Engineering*, pp. CS169-174, Las Vegas, NV.

A. Bivens, R. Gupta, I. McLead, B. Szymanski and J. White, 2002a, ``Scalability and Performance of an Agent-based Network Management Middleware," *International Journal of Network Management*, submitted.

A. Bivens, M. Embrechts, and B.K. Szymanski, 2000a, ``Forecasting and Mitigating Network Congestion using Neural Networks," *5th Online World Conference on Soft Computing in Industrial Applications*,http://wsc-virtual.hut.fi/.

J. Bivens, M. Embrechts, and B.K. Szymanski, 2002b, ``Network Congestion Arbitration and Source Problem Prediction Using Neural Networks," *Smart Engineering System Design*, vol. 4, pp. 243-252.

J. Bivens, B.K. Szymanski, and M. Embrechts, 2000b, ``Network Congestion Arbitration and Source Problem Prediction using Neural Networks," *Proc. Artificial Neural Networks in Engineering, ANNIE'2000*, ASME Press, Fairfield, NJ, 2000, pp.489-494.

J. Branch, A. Bivens, C-Y. Chan, T-K. Lee and B.K. Szymanski, 2002, ``Denial of Service Intrusion Detection Using Time Dependent Deterministic Finite Automata," *Proc. Graduate Research*

*Conference*, Troy, NY, pp. 45-51.

Cannady, J., 1998, "Artificial Neural Networks for Misuse Detection," Proceedings, National Information Systems Security Conference (NISSC'98), October, Arlington, VA, pp. 443-456.

Casilari, E., Alfaro, A., Reyes, A., Diaz-Estrella, A., and Sandoval, F., 1998, "Neural modelling of Ethernet traffic over ATM networks," EANN '98, June, Gibraltar, pp. 304-307.

Cunningham R and Lippmann R , "Improving Intrusion Detection performance using Keyword selection and Neural Networks. " MIT Lincoln Laboratory. http://www.ll.mit.edu/IST/ideval/pubs/pubs_index.html

http://www.fortunecity.com/skyscraper/millenit/262/neural.html

Ghosh A, Schwartzbard A, "A study using Neural Networks for anomaly detection and misuse detection", Reliable Software Technologies.

Girardin, L., and Brodbeck, D., 1998, "A Visual Approach or Monitoring Logs," In Proceedings of the 12th System Administration Conference (LISA '98), Boston, MA, December, pp. 299-308.

Haykin, Simon, 1999, "Neural Networks," Prentice Hall Inc, Upper Saddle River, New Jersey.

Ryan, J., Lin, M., and Mikkulainen,R., 1998, "Intrusion Detection with Neural Networks," Advances in Neural Information Processing Systems, vol. 10, MIT Press.

Stevens, Richard, 1994, "TCP/IP Illustrated Volume 1: The protocols," Addison-Wesley Publishing Company, Reading, Massachusetts, Vol. 1, pp. 12.