10-17-2005

Neural networks to intrusion detection

Dmitry V. Novikov

Neural Networks to Intrusion Detection

Master's Thesis

by Dima Novikov

Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623

Advisor: Leon Reznik
Dr. Leon Reznik — Date

Reader: Hans-Peter Bischof — Nov - 10 - 05
Dr. Hans-Peter Bischof — Date

Observer: Roman Yampolskiy — 11.10.05
Roman Yampolskiy — Date

October 17, 2005

Release Permission Form

Rochester Institute of Technology

Neural Networks to Intrusion Detection

I, Dmitry V. Novikov, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Dmitry V. Novikov 12/8/05

Dmitry V. Novikov Date

Acknowledgments

This work would not have been possible without the support and encouragement of my wife, Lera, over the last two years. Nor would it have been accomplished without valuable efforts to push me towards the completion of this goal by my colleague and friend Roman Yampolskiy. I would like to especially thank my Professors, Dr. Leon Reznik, for his wise advisement and assistance, and Dr. Hans-Peter Bischof, for his insight and help. Thank you all for being there for me!

Table of Contents

List of Figures

List of Tables

Abstract

Recent research indicates a lot of attempts to create an Intrusion Detection System that is capable of learning and recognizing attacks it faces for the first time. Benchmark datasets were created by the MIT Lincoln Lab and by the International Knowledge Discovery and Data Mining group (KDD). A few competitions were held and many systems developed. The overall preference was given to Expert Systems that were based on Decision Making Tree algorithms. This work is devoted to the problem of Neural Networks as means of Intrusion Detection. After multiple techniques and methodologies are investigated, we show that properly trained Neural Networks are capable of fast recognition and classification of different attacks. The advantage of the taken approach allows us to demonstrate the superiority of the Neural Networks over the systems that were created by the winner of the KDD Cups competition and later researchers due to their capability to recognize an attack, to differentiate one attack from another, i.e. classify attacks, and, the most important, to detect new attacks that were not included into the training set. The results obtained through simulations indicate that it is possible to recognize attacks that the Intrusion Detection System never faced before on an acceptably high level.

1 Introduction

The constant development of computer technologies is undeniable. Lots of companies set up shops on the Internet and customers interest in the Web shopping and information gathering is growing rapidly. The Internet became a public tool for communication. Networking has become a very important part of our society. Some devices such as refrigerators, air conditioners, microwaves, and toasters are becoming part of Home Area Networks.

With this incredible growth of technology we are facing an increased need for more security. The accessibility, openness, and complexity of the Internet have provided the need in more sophisticated security of information systems. There is no such company which would be willing its assets stolen, and nobody would like to come home to a burned-down apartment or a house because somebody decided an oven would be fun to crack. Malicious usage, attacks, and sabotage have been on the rise as more and more computers are put into use. Connecting information systems to networks such as the Internet and public telephone systems further magnifies the potential for exposure through a variety of attack channels.

In this research we investigate the problem of Intrusion Detection Systems (IDS), one of very important components of computer/network security. An IDS by itself does not prevent security brakes, but detects malicious use by monitoring unusual activity. This unusual activity is capable of taking an unlimited number of forms.

Most IDS perform monitoring of a system by looking for specific "signatures" of behavior. However, using current methods, it is almost impossible to develop comprehensive-enough databases to warn of attacks. This is for three main reasons. First, these signatures must be hand-coded. Attack signatures that are already known are coded into a database, against which the IDS uses to check current behavior. This system may be imagined as being very rigid. Second, because there is a theoretically infinite number of methods and variations of attacks, an infinite size database would be required to detect all possible attacks. This, of course, is not feasible. Also, any attack that is not included in the database has the potential to cause great harm. One other problem is that current methods are likely to raise many false alarms. So not only do novel attacks succeed, but legitimate use can actually be discouraged.

This work offers a thorough analysis of current IDS models. We study and investigate the benchmarks for this problem provided by the Defense Advanced Research Projects Agency (DARPA) and the International Knowledge Discovery and Data Mining Group (KDD). These benchmarks and the experience of prior researchers are utilized to create an IDS that is capable to learn attack behavior and is able to identify new attacks without system update. In other words, we create a flexible system that does not need hand-coded database of signatures, and that can define new attacks based on pattern, not fixed rules provided by a third party. Neural Networks are chosen as the means of achieving this goal. The use of Neural Networks allows us to identify an attack from the training set, also it allows us to identify new attacks, not included into the training set, and perform attack classification.

2 Neural Networks Overview

As the name implies, neural networks take a cue from the human brain by emulating its structure. Work on neural networks began in the 1940s by McCulloch and Pitts and was followed by the advent of Frank Rosenblatt's Perceptron [2]. The neuron is the basic structural unit of a neural network. In the brain, a neuron receives electrical impulses from numerous sources. If there are enough agonist signals, the neuron fires and triggers all of its outputs. A neural network neuron functions similarly. A neuron receives any number of inputs that possess weights based on their importance. Just as in a real neuron, the weighted inputs are summed and output based on a threshold function sent to every neuron downstream. A barrage of positive inputs will provide a positive output and vice-versa. The original Perceptron received two inputs, and gave a single output. Although this system worked well for simple problems, Minsky demonstrated in 1969 that non-linear classifications, such as exclusive-or (XOR) logic, were impossible [14].

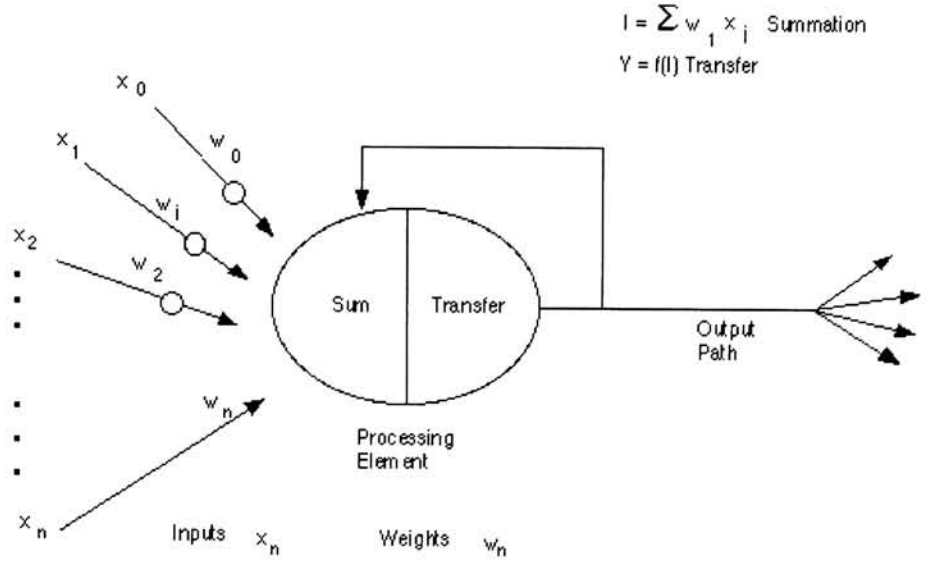

Figure 1. Basic representation of an artificial neuron

In Figure 1, various inputs to the network are represented by the mathematical symbol, x_n. Each of these inputs is multiplied by a connection weight. These weights are represented by w_n. In the simplest case, these products are plainly summed, fed through a transfer function to generate a result, and then output. This process lends itself to physical implementation on a large scale in a small package. This electronic implementation is still possible with other network structures, which utilize different summing functions as well as different transfer functions [14].

It wasn't until the 1980's that training algorithms for multi-layered networks were introduced to solve this problem, restoring faith in neural networks. A multi-layered network consists of numerous neurons, which are arranged into levels. Each level is interconnected with the one above and below it. The first layer receives external inputs and is aptly named the input layer. The top layer provides the classification solution, and is called the output layer. Sandwiched between the input and output layers are any

number of hidden layers. It is believed that a three-layered network can accurately classify any non-linear function. Multi-layered networks commonly use more sophisticated threshold functions, such as the sigmoid function. This is advantageous because the sigmoid function's range is [-0.5, 0.5]; therefore, it prevents any individual output from becoming too large and "overpowering" the network [14].

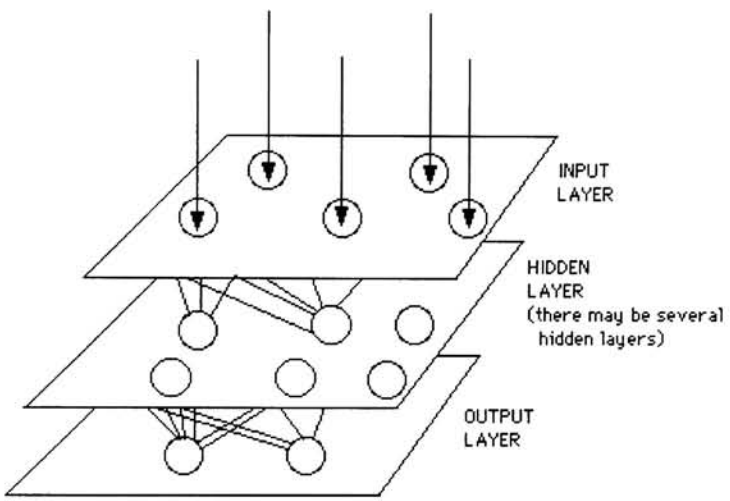

Figure 2. Multi-Layer Perceptron Architecture

Basically, all artificial neural networks have a similar structure or topology as shown in Figure 2. In that structure some of the neurons interface to the real world to receive their inputs. Other neurons provide the real world with the network's outputs. These outputs might be the particular character that the network thinks that it has scanned or the particular image it thinks is being viewed. All the rest of the neurons are hidden from view [14].

2.1 Multi Layer Perceptron Neural Network

This is probably the most popular artificial neural network, which is being used today. We briefly discussed this type of network in previous section: the neurons sum biased, weighted inputs and pass it through a transfer function to produce the output. The neurons are organized in a layered feed-forward topology. Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function complexity. Very significant issues in Multi-layer Perceptron (MLP) are the number of hidden layers and the number of neurons in these layers. The number of input and output neurons is defined by the problem [38].

The most common example of a neural network training algorithm is back propagation. There are modern derived algorithms, such as Levenberg-Marquardt and conjugate gradient descent, are considerably faster in some cases. However under some circumstances back-propagation still has advantages, and is the easiest algorithm to implement and understand. Also, there are some heuristic modifications of back propagation, which work well in certain areas.

In back propagation the gradient vector of the error surface is calculated. This vector points along the line of steepest descent from the current point, so we know that if we move along it a "short" distance, we will decrease the error. A sequence of such moves (slowing as we near the bottom) will eventually find a minimum of some sort. The most challenging part is to decide how large the steps should be. Large steps converge more quickly, but may also overstep the solution or (if the error surface is very eccentric) go

off in the wrong direction. A classic example of this behavior in neural network training is in the progress of the algorithm, which moves very slowly along a steep, narrow, valley, bouncing from one side across to the other. In contrast, very small steps may go in the correct direction, but they require a large number of iterations. In practice, the step size is proportional to the slope (so that the algorithms settles down in a minimum) and to a special constant: the learning rate. The correct setting for the learning rate is application-dependent, which is typically chosen by experiment; it may also be time varying, getting smaller as the algorithm progresses [38].

Back-propagation algorithm can also be modified by the usage of a momentum rate: this guarantees the movement in a set direction. If a few steps are taken in the same path, the algorithm "picks up speed", what makes it possible sometimes to escape local minimum and to shift quickly over flat spots and plateaus. That's why a number of epochs are required for the algorithm to progresses iteratively. During each epoch the training sets are input to the network, then the target and actual outputs are compared, and the error is calculated. The given error is used to modify the weights, and then the algorithm repeats. Initially, the network configuration is random. The training should be stopped in following cases:

- A given number of epochs is reached
- When the error reaches the desired margin
- When the error remains unchanged after completing a certain number of epochs.

2.2 Radial Basis Function Neural Network

Compared to the feed-forward network, the radial basis function (RBF) network is the next-most-used network model. As the name implies, this network makes use of radial functions. Figure 2.1 illustrates an RBF network with inputs $x_1,...,x_n$ and output y. The arrows in the figure symbolize parameters in the network. The RBF network consists of one hidden layer of basis functions, or neurons. At the input of each neuron, the distance between the neuron center and the input vector is calculated. The output of the neuron is then formed by applying the basis function to this distance. The RBF network output is formed by a weighted sum of the neuron outputs and the unity bias [40].

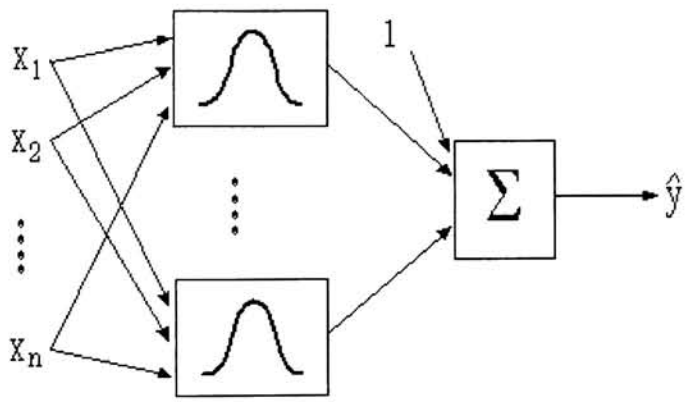

Figure 3. Radial Basis Function Neuron.

MLP neurons are distinct by their weights and threshold, which give the equation of the defining line, and the rate of the function fall-off from that line. Before the sigmoid or other activation function is applied, the activation significance of the neuron is determined through the use of a weighted sum. This sum is the product of the input vector and the weight vector of the neuron.

In contrast, a radial neuron is defined by its center point and a radius. In a radial neuron the center is equivalent to the weights in MLP, and the radius value is stored as the threshold. Therefore, a radial basis function network (RBF) has one hidden layer of neurons, where each neuron includes a Gaussian activation function. Since this function is nonlinear, it is enough to have only one hidden layer. As for the output, it is sufficient to use a linear combination of these outputs (i.e., a weighted sum of the Gaussians) to model any nonlinear function. The standard RBF has an output layer containing neurons with identity activation function [38].

Based on research in this area it is possible to conclude that RBF networks have a number of advantages over MLP:

- They are capable of modeling nonlinear function using a single hidden layer, which disengages design-decisions problems about numbers of layers.

- A linear conversion in the output layer can be optimized using traditional linear modeling techniques, which allow RBF network to train much faster. [41]

2.3 Self-Organizing Map Neural Networks

Whereas all the above-mentioned networks belong to supervised learning, Self Organizing Map Networks (SOM) fits into unsupervised learning. The SOM was developed by Kohonen [27]. While in supervised learning the training data set contains input vector and corresponding output vector, in unsupervised learning the training data set contains only input vector (corresponding output vector does not exist).

At first it is possible to think that it is very odd that the network can learn without outputs – what can it learn? But if we think about it, the answer is pretty simple – SOM learns the structure and the pattern of the data.

Therefore, one potential application for this kind of network is data analysis. The SOM is capable of learning to recognize clusters of data and classify those data. When data are classified and recognized, they can be labeled, so that the network becomes capable of performing classification tasks. SOM networks can be also used for categorization, when output classes are immediately available. In this case, the advantage of SOM is the ability to highlight similarities between classes [38]. Another use of SOM is novelty detection, which is associated with recognizing new, unknown, or unusual behavior.

As for the structure of SOM, they have only two layers: the input layer, and the output layer of radial units, also known as the topological map layer. The units in the topological map layer are laid out in space, typically in two dimensions. Figure 4 demonstrates basic Self Organizing Map, or Kohonen Map architecture.

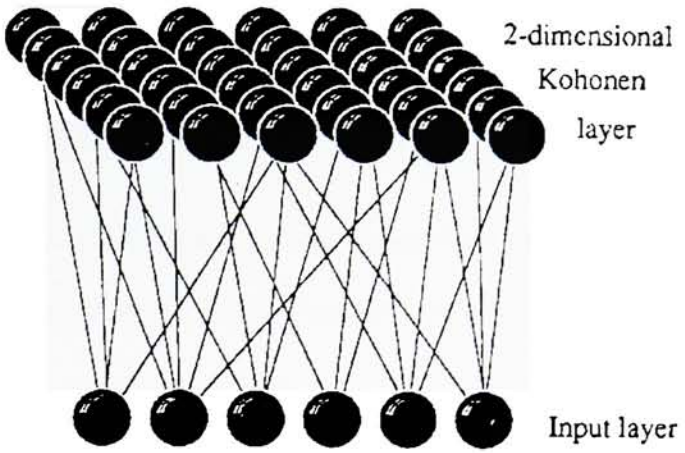

Figure 4. Kohonen Self-Organizing Map Architecture.

An iterative algorithm is used for training SOM networks. Originally, the set of radial centers is initialized randomly. Then, the algorithm gradually adjusts those centers to reflect the clustering of the training data. The iterative training procedure also arranges the network so that neurons representing centers close together in the input space are also located in the same manner on the topological map [14].

The basic iterative Kohonen algorithm simply runs through a number of epochs, on each epoch executing each training case and applying the following algorithm:

- Select the winning neuron, the one, which center is the nearest to the input case, compute a Euclidean Distance between the pairs of neurons, and identify the minimum: $d = \|X - W_j\| = \left[\sum_{i=1}^{n}(x_i - w_{ij})^2\right]^{\frac{1}{2}}$, $Winner = \min_i \|X - W_i\|, i = 1,2,3...m$,

  where x_i and w_{ij} are the elements of the vectors X (input) and W (weights), n is the number of input neurons, and m is the number of neurons in the Kohonen layer.

- Update weights for the winning neuron and it's neighborhood: $\Delta w_i = \alpha(x - w_i)$, where α is a learning rate.

- Iterate until min Euclidean Distance is satisfied, or no noticeable changes in the feature map are found, or the desired number of iterations is reached.

The algorithm uses a time-decaying learning rate, which is set to perform the weighted sum and verifies if the alterations become subtler as the epochs pass. This ensures that the centers settle down to a compromise representation of the cases, which causes the neuron to win.

The neighborhood is a set of neurons near the winning neuron. The neighborhood decreases over time, so that originally it contains a large number of neighboring neurons, perhaps almost the entire topological map. After a number of iterations the neighborhood will contain no neurons (i.e., consists exclusively of the winning neuron itself). Kohonen algorithm adjusts weights not only of the winning neuron, but also the weights of the neurons in its neighborhood.

Once the network has been trained to recognize structure in the data, it can be used as a visualization tool to examine those data. The number of times each neuron wins during the training stage can be counted, what makes it possible to see if distinct clusters have formed on the map. Individual cases are executed and the topological map is observed to see if some meaning could be assigned to the clusters. Once clusters are identified, neurons in the topological map are labeled to indicate their meaning. After the topological map has been built up in this way, new cases can be submitted to the network.

If the winning neuron has been labeled with a class name, the network can perform classification. Otherwise, the network is regarded as undecided [38].

3 Intrusion Detection Overview

Intrusion can be defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource. A complete guidance document on Intrusion Detection Systems is available at the National Institute of Standards and Technology (NIST) [4]. Effective intrusion detection is a difficult and elusive goal for system administrators and information security researchers. The inherent complexity of computer systems, the variety of potential vulnerabilities, and the skills of many attackers combine to create a problem domain that is extremely difficult to address [36].

In the context of information systems, intrusion refers to any unauthorized access, not permitted attempt to access or damage, or malicious use of information resources. Intrusion can be categorized into two classes: anomaly intrusions and misuse intrusions [34]. Thus, intrusion detection has traditionally focused on one of two approaches: anomaly detection or misuse detection.

Anomaly detection seeks to identify activities that vary from established patterns for users, or groups of users. It typically involves the creation of knowledge bases compiled from profiles of previously monitored activities. Anomaly detection is usually achieved through one of the following

1) Threshold detection, detecting abnormal activity on the server or network, for example abnormal consumption of the CPU for one server, or abnormal saturation of the network.

2) Statistical measures, learned from historical values.

3) Rule-based measures, with expert systems.

4) Non-linear algorithms such as Neural Networks or Genetic Algorithms [35].

The second approach, misuse detection, compares a user's activities with the known behaviors of attackers attempting to penetrate a system. Anomaly detection often uses threshold monitoring to identify incidents, while misuse detection is most often accomplished using a rule-based approach. The misused detection is usually achieved through one of the following:

1) Expert systems, containing a set of rules that describe attacks.

2) Signature verification, where attack scenarios are translated into sequences of audit events.

3) Petri nets, where known attacks are represented with graphical Petri nets.

4) State-transition diagrams, representing attacks with a set of goals and transitions [35].

Expert systems are the most common form of rule-based intrusion detection approaches. Unfortunately, expert systems have little or no flexibility; even minor variations in an attack sequence can affect the activity-rule comparison to a great enough degree to prevent detection. Some approaches have increased the level of abstraction of the rule-

based approach in an attempt to compensate for this weakness, with a side effect of reducing the granularity of the intrusion detection process [36].

The most common method to identify intrusions is the method, which makes use of the log data generated by special software, like firewalls, or the operating system. It is possible that a manual examination of those logs would make it sufficient to detect intrusions. Analyzing the data even after an attack has taken place to decide the degree of damage sustained is trivial. This examination also plays a significant role in tracking down the intruders and recording the attack patterns for future detections. A well designed IDS that can be used to analyze audit data for such insights makes a valuable tool for information systems. The idea behind anomaly detection is to establish each user's normal activity profile, and to flag deviations from the established profile as possible intrusion attempts. A main issue concerning misuse detection is the signature development that includes all possible attacks to avoid false negatives, and the signature development that does not match non-intrusive activities to avoid false positives. Though, false negatives are frequently considered more serious. The selection of threshold levels is important, so that neither of the above problems is unreasonably magnified [34].

A number of IDS commercial tools are widely available for security specialists nowadays. Most of them deal with the misuse detection model. The main problem with such systems is their low flexible, as it was mentioned before. So, they are not capable of detecting new, unknown, novel behavior, and require constant update by vendors.

The US Defense Advanced Research Projects Agency (DARPA) has sponsored a large-scale realistic Intrusion Detection database in 1998 to improve performance of IDS

systems with real network traffic. More than two months of traffic observed from the US Government sites and the Internet was registered, adding attacks against various hosts OS. DARPA database was designed to evaluate performances of Intrusion Detection Systems. The first evaluation with offline and real time database was conducted in the summer of 1998 [15].

4 Artificial Intelligence Techniques in Intrusion Detection

Classical network based approaches to the problem of detection network borne intrusions often rely on either rule-based misuse detection or anomaly detection [5]. Rule-based misuse detection systems attempt to recognize specific behaviors that represent known forms of abuse or intrusion. Typically, they require an exhaustive list of templates characterizing each attack instance; there is no concept of similarity to a currently listed attack instance. On the other hand, anomaly detection attempts to recognize abnormal user behavior. Actually, it identifies "normal" behaviors by mining the monitored behavior of each user so that "abnormal" behaviors can be characterized.

One of the most interesting researches in the area of Intrusion Detection deals with the applications of the Artificial Intelligence (AI) techniques. The most valuable feature of any AI system is the ability to learn automatically according to data inputs and outputs. This characteristic potentially can add more flexibility to Intrusion Detection Systems and remove the necessity to update the database of possible attacks constantly. At this point we talk not just about an IDS, but about Intelligent Intrusion Detection System (IIDS), which is capable of creating attack patterns, i.e. learning about new attacks, based on previous experience.

Multiple experiments were held to apply Artificial Techniques in Intrusion Detection. The main goal was to create a system that is capable of detecting different kinds of attacks. The researchers used DARPA [15] and KDD Cups [26] benchmark databases for

training and detecting attacks in the experiments. In this chapter we discuss the approaches the authors chose to detect and/or classify attacks and the results they have obtained.

4.1 Neural Networks Approach

Neural Networks approach is one of the most interesting in this area. An increasing amount of research in the last few years has investigated the application of neural networks to intrusion detection. If properly designed and implemented, neural networks have the potential to address many of the problems encountered by rule-based approaches. Neural networks were specifically proposed to learn the typical characteristics of systems users and identify statistically significant variations from their established behavior. In order to apply this approach to Intrusion Detection, we would have to introduce data representing attacks and non-attacks to the Neural Network to adjust automatically coefficients of this Network during the training phase. In other words, it will be necessary to collect data representing normal and abnormal behavior and train the Neural Network on those data. After training is accomplished, a certain number of Performance tests with real network traffic and attacks should be conducted.

Supervised Learning Model

Lippmann and Cunningham of MIT Lincoln Laboratory conducted a number of tests employing Neural Networks to misuse detection [12, 13]. The system was searching for attack-specific keywords in the network traffic. A Multi-layer Perceptron had been used for detection UNIX host attacks, and attacks to obtain root-privilege on a server. The system was trying to detect the presence of an attack by classifying the inputs into 2 (two) outputs (normal and attack). The system was able to detect 80% of attacks - 17 out of 20

attacks were identified. The main achievement of this system was its ability to detect old as well as new attacks not included in the training data.

Unsupervised Learning Model

There were a couple of systems, which used Self Organizing Maps for detecting intrusion. Luc Girardin's of UBILAB laboratory performed clustering of network traffic in order to detect attacks. A visual approach was chosen for attack association [21]. SOM were employed to project network events on an appropriate 2D-space for visualization, then the network administrator analyzed them. Intrusions were extracted from the view by highlighting divergence from the norm with visual metaphors of network traffic. The main disadvantage of this approach is its need in interpretation of network traffic by an administrator or other authorized person to detect attacks.

Kayacik, Zincir-Heywood, and Heywood utilize KDD Cups data set for the experiments. They create three layer of employment [25]:

1) First, individual SOM are associated with each basic TCP feature. This provides a concise summary of the interesting properties of each basic feature, as derived over a suitable temporal horizon.

2) Second, integrates the views provided by the first level SOM into a single view of the problem. At this point, they use the training set labels associated with each pattern to label the respective best matching unit in the second layer.

3) Third, the final layer is built for those neurons, which win for both attack and normal behaviors. This results in third layer SOMs being associated with specific neurons in the second layer. Moreover, the hierarchical nature of the architecture means that the first layer may be trained in parallel and the third layer SOMs are only trained over a small fraction of the data set.

	Normal	*DoS*	*Probe*	*U2R*	*R2L*
Level 2	92.4	96.5	72.8	22.9	11.3
Level 1	95.4	95.1	64.3	10.0	9.9

Table 1. Performance of 2 and 3 layer hierarchy on different categories

The table above describes the detection of attacks by category performed by the authors. They also performed individual attack detection. In the table below we can see the individual attack detection rates.

Attack Name	*Level 2*	*Level 3*
Apache2	90.3	90.7
Httptunnel	58.9	20.9
Mailbomb	7.8	6.8
Mscan	90.2	60.9
Named	23.5	0.0
Processtable	59.4	47.6
Ps	0.0	0.0
Saint	79.1	78.7
Sendmail	5.9	11.8
Snmpgetattack	11.5	10.3
Udpstorm	0.0	0.0
Xlock	0.0	0.0
xsnoop	0.0	0.0
Xterm	23.1	30.8

Table 2. Detection rate of new attacks for 2-layer and 3-layer hierarchy

Hybrid Networks

Several researchers have combined MLP and SOM in their attempt to create an Intrusion Detection System. Cannady and Mahaffey of Georgia Technical Research Institute and Fox, Henning, and Reed have performed a research to apply Multi-Layer Perceptron model and Self-Organizing Map for misuse detection [10, 19, 35]. They have used a feed-forward network with back-propagation learning, which contained 4 fully connected layers, 9 input nodes and 2 output nodes (normal and attack). The network has been trained for a certain number of attacks. The network has succeeded in identifying attacks it was trained for.

Bivens believes that DOS and other network-based attacks leave a faint trace of their presence in the network traffic data. He has designed modular network-based intrusion detection system that analyzes TCP dump data to develop windowed traffic intensity trends, which detects network-based attacks by carefully analyzing this network traffic data and alerting administrators to abnormal traffic trends. It has been shown that network traffic can be efficiently modeled using artificial neural networks [3, 12, 13], therefore MLP was chosen to examine network traffic data. SOM has been used to group network traffic together to present it to the neural network, as SOM have been shown to be effective in novelty detection [21, 25, 43].

The data that they have presented to the neural network consisted of attack-specific keyword counts in network traffic [32]. This system reminds a host-based detection system because it looks at the user actions. The Neural Network was created to analyze program behavior profiles instead of user behavior profiles [20]. This method identifies

the normal system behavior of certain programs and compares it to the current system behavior. The author has used DARPA benchmark for the experiments. The prediction rate of the system is 24% - 100%. 100% has been achieved only with one attack in the training set – sshprocesstable [7, 9].

4.2 Rule-Based Approach

Agarwal and Joshi propose a two-stage general-to-specific framework for learning a rule-based model (PNrule) to learn classifier models on a data set that has widely different class distributions in the training data [1]. They utilized KDD Cups database for training and testing their system. The system was classifying the attacks into 4 main groups:

- Probing – information gathering
- Denial of Service (DOS) – deny legitimate requests to the system
- User-to-Root (U2R) – unauthorized access to local super-user or root
- Remote-to-Local (R2L) – unauthorized local access from a remote machine

As the result, the system performed very well on detecting Probing and DOS attacks identifying 73.2% and 96.6% respectively. 6.6% of U2R attacks were detected and 10.7% of R2L. False alarms were generated at a level of less than 10% for all attack categories except for U2R – an unacceptably high level of 89.5% false alarm rate was reported for this category

4.3 Decision-Tree Approach

Levin creates a set of locally optimal decision trees (decision forest) from which optimal subset of trees (sub-forest) is selected for predicting new cases [31]. 10% of KDD Cups database is used for training and testing. Data is randomly sampled from the entire training data set. Multi-class detection approach is used to detect different attack categories in the KDD data set. Just like Agarwal and Joshi [1], Levin tries to classify the data into four main categories: Probing, DOS, U2R, and R2L. The final trees give very high detection rates for all classes including the R2L in the entire training data set. In particular, 84.5% detection rate for Probing, 97.5% for DOS, 11.8% for U2R, and 7.32% for R2L. The following false alarm rates were detected for Probing, DOS, U2R and R2L attack categories respectively - 21.6%, 73.1%, 36.4%, and 1.7%.

4.4 Shared Nearest Neighbor and K-Means Approach

Ertoz used shared nearest neighbor technique (SNN) that is particularly suited for finding clusters in data of different sizes, density, and shapes, mainly when the data contains large amount of noise and outliers. All attack records are selected from the KDD training and testing data sets with a count of 10,000 records from each attack type: there are a total of 36 attack types from 4 attack categories. Also, 10,000 records were randomly picked from both the training and the testing data sets. In total, around 97,000 records were selected from the entire KDD data set. After removing duplicate KDD records, the data set size reduced to 45,000 records [18]. The author is utilizing two main clustering algorithms: K-Means, where the number of clusters is equal to 300, and SNN. K-Means performed very well on Probing, DOS, and R2L, detecting 91.8%, 98.75%, and 77.04% respectively. Detection rate for U2R is 5.6%. SNN performed in the following manner: 73.48% for Probing, 77.76% for DOS, 37.82% for U2R, and 68.15% for R2L. False alarms are not discussed by the author.

4.5 Parzen-Window Approach

Yeung and Chow propose a novelty detection approach using non-parametric density estimation based on Parzen-window estimators with Gaussian kernels to build an intrusion detection system using normal data only. This novelty detection approach was employed to detect attack categories in the KDD data set [42]. 30,000 randomly sampled normal records from the KDD training data set were used as training data set to estimate the density of the model. 30,000 randomly sampled normal records (also from the KDD training data set) formed the threshold determination set, which had no overlap with the training data set. The results were very high in most cases: 99.17% detection of Probing, 96.71% of DOS, 93.57% of U2R, and 31.17% of R2L. No false alarms information is available. The main advantage of this technique is its capability of detecting an attack, not just classifying the attacks.

4.6 Multi-Classifier Approach

Sabhnani and Serpen of the University of Toledo conduct a number of experiments with hybrid systems that contained different approaches for attack classification. KDD Cups database is chosen for the experiments. The attacks are classified into four main groups, as was done by the researchers discussed in prior chapters:

- Probing – information gathering
- Denial of Service (DOS) – deny legitimate requests to the system
- User-to-Root (U2R) – unauthorized access to local super-user or root
- Remote-to-Local (R2L) – unauthorized local access from a remote machine

They highlight that most researchers employ a single algorithm to detect multiple attack categories with dismal performance in some cases. So, they propose to use a specific detection algorithm that is associated with an attack category for which it is the most promising [37].

Attributes in the KDD datasets had all forms – continuous, discrete, and symbolic, with significantly varying resolution and ranges. Most pattern classification methods are not able to process data in such a format. Hence, preprocessing was required. The preprocessing includes the following steps:

1) Mapping symbolic-valued attributes to numeric-valued attributes. Symbolic features like protocol_type (3 different symbols), service (70 different symbols), and flag (11 different symbols) were mapped to integer values ranging from 0 to N-1 where N is the number of symbols. Attack names, such as buffer_overflow, were mapped to one of the five classes:

   a. Normal was mapped to 0

   b. Probing was mapped to 1

   c. DOS was mapped to 2

   d. U2R was mapped to 3

   e. 4 – R2L as described in [17]

2) Scaling.

   a. Each of the mapped features was linearly scaled to the range [0.0, 1.0].

   b. Features having smaller integer value ranges like duration [0, 58329], wrong_fragment [0, 3], urgent [0, 14], hot[0, 101], num_failed_logins [0, 5], num_compromised [0, 9], su_attemptes [0, 2], num_root [0, 7468], num_file_creations [0, 100], num_shells [0, 5], num_access_files [0, 9], count [0, 511], srv_count [0, 511], dst_host_count [0, 255], and

dst_host_srv_count [0, 255] were also scaled linearly to the range of [0, 1].

c. Logarithmic scaling (base 10) was applied to two features spanned over a very large integer range, namely src_bytes [0, 1.3 billion] and dts_bytes [0, 1.3 billion], to reduce the range to [0, 9.14].

d. All other features were either Boolean, like logged_in, having values (0 or 1), or continuous, like diff_srv_rate, in the range of [0, 1]. No scaling was necessary for these attributes

3) For the purpose of training different classifier models, all duplicate records are removed from the datasets. The total number of records in the original labeled training dataset is 972, 780 for normal; 41, 102 for Probe; 3,883,370 for DOS; 52 for U2R and 999 for R2L attack classes.

All simulations are performed on a multi-user Sun SPARC machine, which has dual microprocessors, ULTRASPARC-II, running at 400 MHz. System clock frequency is equal to 100 MHz., the system had 512 MB of Ram and Solaris 8 operating system.

9 distinct pattern recognition and machine learning algorithms are tested:

1) MLP. 3 layers of Feed Forward Neural Network are implemented. Sigmoid is used as the transfer function and stochastic gradient decent with mean squared error function as the learning algorithm. The network has 41 (forty one) inputs; 5 outputs; 40 – 80 nodes in the hidden layer; 0.1 – 0.6 learning rate (0.1 the final

rate); 500,000 samples in each epoch; and 30 – 150 epochs (60 the final number of epochs).

2) Gaussian classifier (GAU). This classifier assumes inputs are uncorrelated and distributions for different classes differ only in mean values. It is based on the Bayes decision theorem [16].

3) K-means clustering (K-M). This algorithm [16] positions K centers in the pattern space such that the total squared error distance between each training pattern and the nearest center is minimized.

4) Nearest cluster algorithm (NEA). It is a condensed version of K-nearest neighbor clustering algorithm [16]. Input to this algorithm is a set of cluster centers generated from the training data set using standard clustering algorithms like K-means, E & M binary split, and leader algorithm. In this case, the initial clusters were created using the K-means.

5) Incremental Radial Basis Function (IRBF). Can perform non-linear mapping between input and output vectors similar to RBF and MLP [22].

6) Leader algorithm (LEA). LEA partitions are a set of M records into K disjoint clusters (where $M \geq K$) [23]. First input record forms the leader of the first cluster. Each input record is sequentially compared with current leader clusters. If the distance measure between the current record and all leader records is greater than

the threshold (delta), a new cluster is formed with the current record being the cluster leader.

7) Hyper sphere algorithm (HYP). This algorithm creates decision boundaries using spheres in input feature space [6] [30]. Any pattern that falls within the sphere is classified in the same class as that of the center pattern. Spheres are created using an initial defined radius. Euclidian distance between a pattern and sphere centers is used to test whether a pattern falls in one of the current defined spheres.

8) Fuzzy ARTMAP (ART). Adaptive Resonance Theory mapping algorithm is used for supervised learning of multidimensional data [11]. It uses two ART's – ARTa and ARTb. ARTa maps features into clusters. ARTb maps output categories into clusters. There is a mapping from ARTa clusters to ARTb clusters that is performed during training.

9) C4.5 decision tree (C4.5). This algorithm was developed by Quinlan [39]. It generates decision trees using an information theoretic methodology. The goal is to construct a decision tree with minimum number of nodes that gives least number of misclassifications on training data. Divide and conquer strategy is utilized in this algorithm. The publicly available pattern classification software tool LNKnet is used to simulate pattern recognition and machine learning models [33]. The C4.5 algorithm is employed to generate decision trees using the software tool obtained at [8].

		Probe	DoS	U2R	R2L
MLP	*PD*	88.7%	97.2%	13.2%	5.6%
	FAR	0.4%	0.3%	0.1%	0.1%
GAU	*PD*	90.2%	82.4%	22.8%	0.1%
	FAR	11.3%	0.9%	0.5%	0.1%
K-M	*PD*	87.6%	97.3%	29.8%	6.4%
	FAR	2.6%	0.4%	0.4%	0.1%
NEA	*PD*	88.8%	97.1%	2.2%	3.4%
	FAR	0.5%	0.3%	0.1%	0.1%
RBF	*PD*	93.2%	73%	6.1%	5.9%
	FAR	18.8%	0.2%	0.4%	0.3%
LEA	*PD*	83.8%	97.2%	8.3%	1%
	FAR	0.3%	0.3%	0.3%	0.1%
HYP	*PD*	84.8%	97.2%	8.3%	1%
	FAR	0.4%	0.3%	0.1%	0.1%
ART	*PD*	77.2%	97%	6.1%	3.7%
	FAR	0.2%	0.3%	0.1%	0.1%
C4.5	*PD*	80.8%	97%	1.8%	4.6%
	FAR	0.7%	0.3%	0.1%	0.1%

Table 3. Multi – Classifier Results for separate algorithms

In the table above we can see the results of the experiments held by the authors. Here PD represents Probability of Detection, and FAR – False Alarm Rate. They state that the set of pattern recognition and machine learning algorithms tested on the KDD data sets offers an acceptable level of misuse detection performance for only two attack categories - Probing and DOS. On the other hand, all nine classification algorithms fail to demonstrate an acceptable level of detection performance for the remaining two attack categories that are U2R and R2L. Thus, [37] offers a multi-classifier model: they propose to have sub-classifiers trained using different algorithms for each attack category. They offer the best algorithm for each category:

1) MLP for probing

2) K-M for DOS

3) K-M for U2R

4) GAU for R2L

The results are depicted in the table below, where we can see the improvement in the performance of the system overall.

		Probe	**DoS**	**U2R**	**R2L**
Multi-Classifier	*Pos Detection*	88.7%	97.3%	29.8%	9.6%
	False Alarms	0.4%	0.4%	0.4%	0.1%

Table 4. Multi-Classifier Results for the final system

5 Experiments

At this point we have accomplished the investigation of the related research and have compared the results achieved by those researchers. Figure 5 demonstrates the workflow of the process of this effort that requires learning the problem, collecting of the data/benchmarks with known results from other researcher, and the experiments conduction. At this stage we have finished researching the problem and managed to get the data that was used by the previous investigators. Now, knowing the results of previous work, we need to prepare the data and hold the experiments. We have come close to the last two stages that are considered to be the most interesting and challenging.

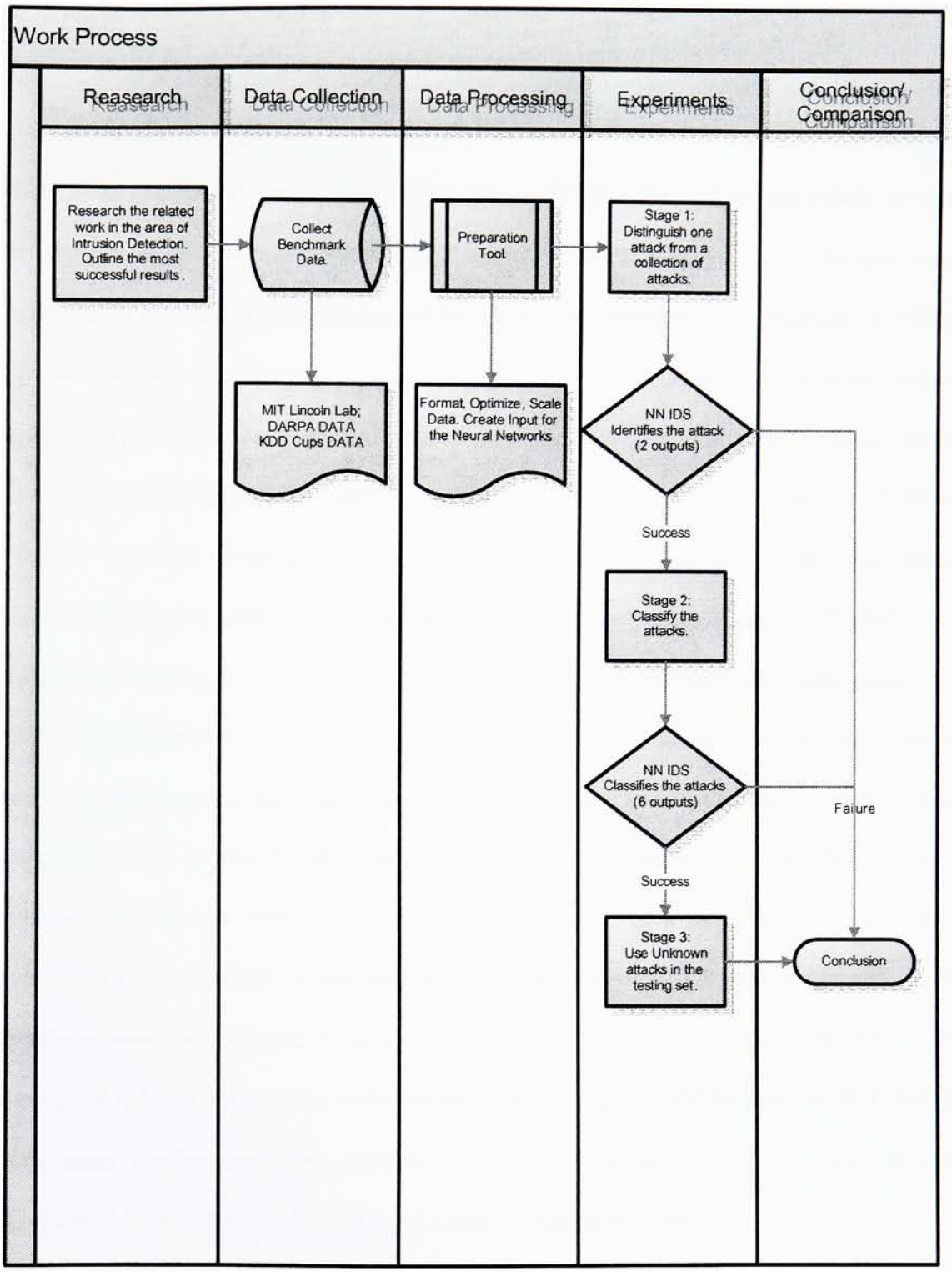

Figure 5. Process Workflow.

5.1 Data

To conduct the experiments, it was decided to use the benchmarks of the International Knowledge Discovery and Data Mining group (KDD). These data are based on the benchmark of the Defense Advanced Research Projects Agency (DARPA) that was collected by the Lincoln Laboratory of Massachusetts Institute of Technology in 1998, and was the first initiative to provide designers of Intrusion Detection Systems with a benchmark, on which to evaluate different methodologies [15]. In order to collect these data, a simulation had been made of a factitious military network consisting of three "target" machines running various operating systems and services. Additional three machines were then used to spoof different IP addresses, thus generating traffic between different IP addresses. Finally, a sniffer was used to record all network traffic using the TCP dump format. The total simulated period was seven weeks. Normal connections were created to profile that expected in a military network and attacks fall into one of five categories: User to Root (U2R), Remote to Local (R2L), Denial of Service (DOS), Data, and Probe. Packets information in the TCP dump files were summarized into connections. Specifically, a connection was a sequence of TCP packets starting and ending at some well defined times, between which data flows from a source IP address to a target IP address under some well defined protocol [24]. In 1999 the original TCP dump files were preprocessed for utilization in the IDS benchmark of the International Knowledge Discovery and Data Mining Tools Competitions [24].

Format

The data consists of a number of basic features:

1. Duration of the connection.

2. Protocol type, such as TCP, UDP or ICMP.

3. Service type, such as FTP, HTTP, Telnet.

4. Status flag.

5. Total bytes sent to destination host.

6. Total bytes sent to source host.

7. Whether source and destination addresses are the same or not.

8. Number of wrong fragments.

9. Number of urgent packets.

In addition to the above nine "basic features", each connection is also described in terms of an additional 32 derived features, falling into three categories:

1. Content features: Domain knowledge is used to assess the payload of the original TCP packets. This includes features such as the number of failed login attempts.

2. Time-based traffic features: these features are designed to capture properties that mature over a 2 second temporal window. One example of such a feature would be the number of connections to the same host over the 2 second interval.

3. Host-based Traffic features: utilize a historical window estimated over the number of connections – in this case 100 – instead of time. Host based features are therefore designed to assess attacks, which span intervals longer than 2 seconds.

Each record consists of 41 attributes and one target [28, 29]. The target value indicates the attack name (Figures 6 and 7).

1,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.0
0,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal.

14,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.
00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattack.

0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.0
0,0.00,0.00,255,254,1.00,0.01,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattack.

0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.0
0,0.00,0.00,255,255,1.00,0.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattack.

0,udp,domain_u,SF,29,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,0.00,0.00,0.00,0.00,0.50
,1.00,0.00,10.3,0.30,0.30,0.30,0.00,0.00,0.00,0.00,0.00,0.00,normal.

0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.0
0,0.00,0.00,255,253,0.99,0.01,0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal.

Figure 6. Preformatted Data Sample

- 01 duration: continuous.
- 02 protocol_type: symbolic.
- 03 service: symbolic.
- 04 flag: symbolic.
- 05 src_bytes: continuous.
- 06 dst_bytes: continuous.
- 07 land: symbolic.
- 08 wrong_fragment: continuous.
- 09 urgent: continuous.
- 10 hot: continuous.
- 11 num_failed_logins: continuous.
- 12 logged_in: symbolic.
- 13 num_compromised: continuous.
- 14 root_shell: continuous.
- 15 su_attempted: continuous.
- 16 num_root: continuous.
- 17 num_file_creations: continuous.
- 18 num_shells: continuous.
- 19 num_access_files: continuous.
- 20 num_outbound_cmds: continuous.
- 21 is_host_login: symbolic.
- 22 is_guest_login: symbolic.
- 23 count: continuous.
- 24 srv_count: continuous.
- 25 serror_rate: continuous.
- 26 srv_serror_rate: continuous.
- 27 rerror_rate: continuous.
- 28 srv_rerror_rate: continuous.
- 29 same_srv_rate: continuous.
- 30 diff_srv_rate: continuous.
- 31 srv_diff_host_rate: continuous.
- 32 dst_host_count: continuous.
- 33 dst_host_srv_count: continuous.
- 34 dst_host_same_srv_rate: continuous.
- 35 dst_host_diff_srv_rate: continuous.
- 36 dst_host_same_src_port_rate: continuous.
- 37 dst_host_srv_diff_host_rate: continuous.
- 38 dst_host_serror_rate: continuous.
- 39 dst_host_srv_serror_rate: continuous.
- 40 dst_host_rerror_rate: continuous.
- 41 dst_host_srv_rerror_rate: continuous.

Figure 7. Data Fields.

Optimization

In order to perform formatting and optimization of the data, a tool was written that is capable of completing such operations as computing data statistics, data conversion, data optimization, neural network input creation, and other data preparation related assignments.

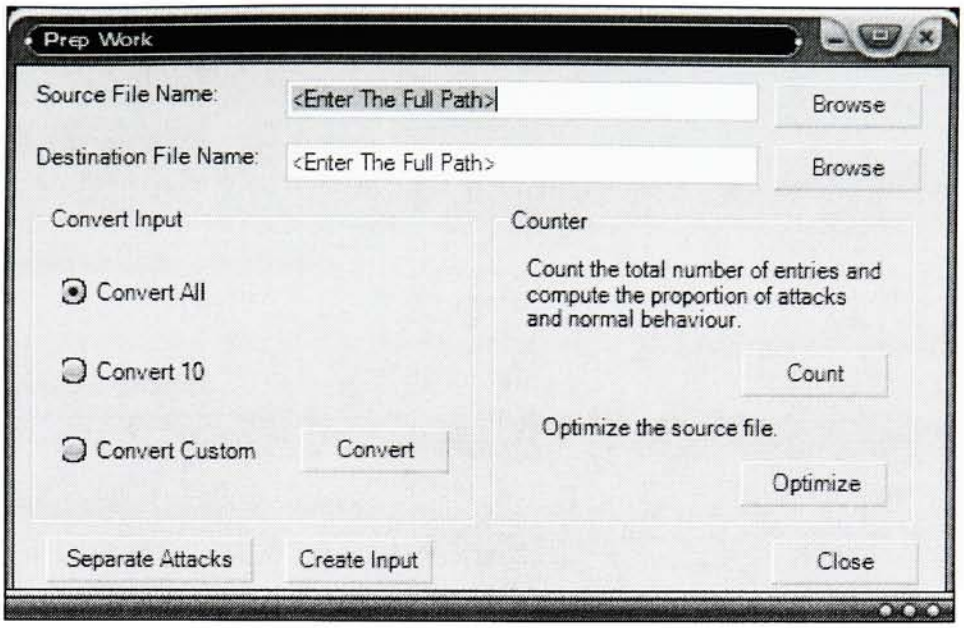

Figure 8. Data Preparation Tool

Based on the results produced by the Preparation Tool, we made the following classifications:

- Each record consists of 41 attributes and one target. The target value indicates the attack name.
- The data has 4,898,431 records in the dataset.
- 3,925,650 (80.14%) records represent attacks that fall into one of the five mentioned above categories.
- Total 22 attacks were identified.
- 972,781 (19.85%) records of normal behavior were found.

The following attacks were identified:

1. Buffer overflow – 30 records.
2. Load module – 9 records.
3. Perl – 3 records.
4. Neptune – 1,072,017 records.
5. Smurf – 2,807,886 records.
6. Guess Password – 53 records.
7. Pod – 264 records.
8. Teardrop – 979 records.
9. Port sweep – 10,413 records.
10. IP sweep – 12, 481 records.
11. Land – 21 records.
12. FTP Write – 8 records.
13. Back – 2,203 records.

14. IMAP – 12 records.

15. Satan – 15,892 records.

16. PHF – 4 records.

17. NMAP – 2,316 records.

18. Multihop – 7 records.

19. Warezmaster – 20 records.

20. Warezclient – 1020 records.

21. Spy – 2 records.

22. Rootkit – 10 records.

Figure 9 shows the graphical representation of these attacks in the dataset. As it is possible to see, the attacks are not distributed evenly. Some attacks, such as Smurf, are represented in a bigger number and occupy more than 50% of the dataset.

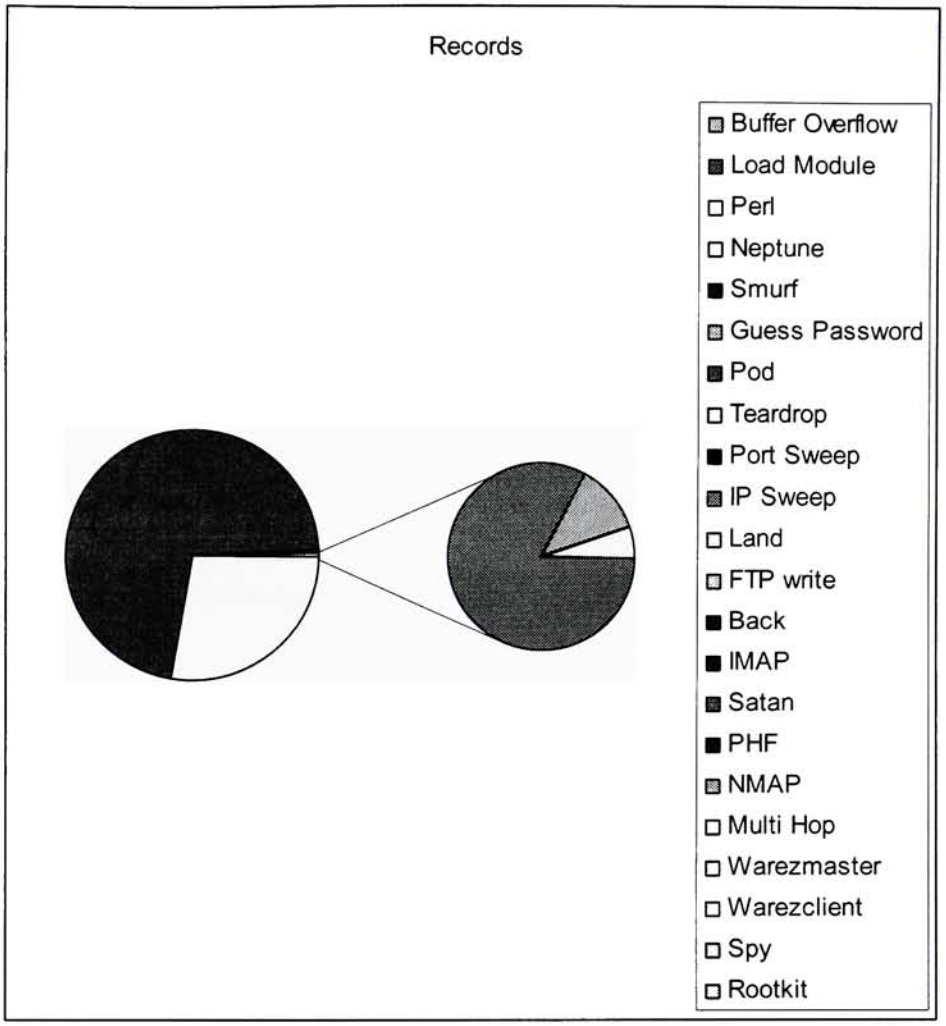

Figure 9. KDD Data.

Attributes in the KDD datasets have all forms – continuous, discrete, and symbolic, with significantly varying resolution and ranges. Most pattern classification methods are not able to process data in such a format. Therefore, preprocessing is required to transform the data into the most optimal format acceptable by the neural networks.

First of all, the dataset was split into multiple files. Each file contained records corresponding to a certain attack or normal behavior. Thus, a library of attacks was

created. It was done to achieve an efficient way to format, optimize, and compose custom training and testing datasets.

Second, symbolic features like attack name (23 different symbols), protocol type (three different symbols), service (70 different symbols), and flag (11 different symbols) were mapped to integer values ranging from 0 to N-1 where N is the number of symbols.

Third, a certain scaling had taken place:

- Each of the mapped features was linearly scaled to the range [0.0, 1.0].

- Features having integer value ranges like duration, wrong_fragment, urgent, hot, num_failed_logins, num_compromised, su_attemptes, num_root, num_file_creations, num_shells, num_access_files, count, srv_count, dst_host_count, dst_host_srv_count, src_bytes, and dts_bytes were also scaled linearly to the range of [0, 1].

- All other features were either Boolean, like logged_in, having values (0 or 1), or continuous, like diff_srv_rate, in the range of [0, 1]. No scaling was necessary for these attributes.

Fourth, for the purpose of efficient training, all duplicate records were removed from the dataset.

Final Data Set

Figure 10 shows the final data set that was processed by the means of the Preparation Tool. All string values were mapped, integer values were scaled in the range of [0, 1], and all Boolean values remain untouched. Therefore, the dataset was prepared for the use by the Neural Networks.

- 0, 0, 0, 0, 0.00181, 0.0545, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.008, 0.008, 0, 0, 0, 0, 0.1, 0, 0, 0.009, 0.009, 0.1, 0, 0.011, 0, 0, 0, 0, 0
- 0, 0, 0, 0, 0.00239, 0.00486, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.008, 0.008, 0, 0, 0, 0, 0.1, 0, 0, 0.019, 0.019, 0.1, 0, 0.005, 0, 0, 0, 0, 0
- 0, 0, 0, 0, 0.00235, 0.01337, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.008, 0.008, 0, 0, 0, 0, 0.1, 0, 0, 0.029, 0.029, 0.1, 0, 0.003, 0, 0, 0, 0, 0
- 0, 0, 0, 0, 0.00219, 0.01337, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.006, 0.006, 0, 0, 0, 0, 0.1, 0, 0, 0.039, 0.039, 0.1, 0, 0.003, 0, 0, 0, 0, 0

Figure 10. Final Dataset.

After the data formatting and optimization, a set of attacks that would be used in the experiments was determined. The optimization process significantly decreased the number of records for each of the attacks, thus, such attacks as RootKit, Spy, Multihop, IMAP, etc do not have sufficient number of records for neither training the neural networks, nor for testing, hence, all attacks with statistically insignificant amount of records were omitted.

Attacks with the most number of records were chosen to be in the training set. The following attacks were used to train and to test the neural networks:

- Smuf
- Satan
- Neptune
- Ipsweep
- Back

The following attacks were chosen for the unknown (not trained) set of attacks:

- Buffer_overflow
- Guess_password
- Nmap
- Teardrop
- Warezclient

5.2 Neural Networks Based Intrusion Detection System Experiments

It was decided to run the experiments in three stages. In stage one, it was important to repeat the experiments of other researchers and have the Neural Networks to identify an attack. In stage two the assignment was set to a more complicated goal. It was decided to classify the attacks, thus, the Neural Networks had to determine not only the presence of an attack, but the attack itself. Stage three had to repeat the experiments of stage two, but in this stage a set of unknown attacks are added to the testing set. Stage three contains experiments of a higher complexity and interest. It should be noted that the normal behavior records are viewed as a "normal" attack. MatLab was chosen to run simulations with Neural Networks.

RBF

Each RBF Neural Network had 41 inputs, corresponding to each attribute in the dataset, two outputs for attack detection (the first output for the presence of an attack – "YES", the second output for the normal behavior – "NO"), or six outputs for attack classification (five outputs for the attacks, and the sixth output for the normal behavior), three layers (input, hidden, and output). The training set consisted of 4000 records. The attack and the normal behavior records were evenly distributed in the training set.

MLP

The parameters of the MLP Neural Network were as follows:

- 41 inputs, corresponding to each attribute in the dataset

- two outputs for attack detection (the first output for the presence of an attack – "YES", the second output for the normal behavior – "NO"), or six outputs for attack classification (five outputs for the attacks, and the sixth output for the normal behavior)

- three layers (input, hidden, and output)

    - the hidden layer has 20 nodes

- alpha = 0.7

- beta = 0.8

- "tansig" function is used in the input layer node, "purelin" in the hidden and output layer nodes

- 50 epochs

- The training set consisted of 4000 records. The attack and the normal behavior records were evenly distributed in the training set.

Before the parameters were set for the Multiple Layer Perceptron, it was optimized. We tried 5 – 150 nodes in the hidden layer. 20 nodes had the best performance, thus it was decided to limit the number of nodes to this amount. Alpha and beta rates were tested in the range of 0 – 10. Multiple activation functions were tried in different layers, "tansig" and "purelin" were chosen as the best fitting ones. Number of epochs was limited to 50, since the network's performance was decreasing significantly with more epochs.

5.3 Results

First Stage - Known Attacks Detection

The first stage of the experiments consisted of 2 phases. At this stage only one attack was used in the training set. The distribution of an attack and normal records was 50% - 50%. Table 5 and Figure 11 represent the results of these experiments. As it is shown, the accuracy of positive recognition is very high for both Neural Networks. All of the attacks have more than 90% of recognition. Most of them are very close to 100%, what is a very good expected result.

Attack Name	RBF Accuracy	RBF False Alarms	MLP Accuracy	MLP False Alarms
Smurf	100%	0	99.5%	0
Neptune	100%	0	100%	0
Satan	91%	7%	97.2%	2%
IP Sweep	99.5%	0	99.9%	0
Back	100%	0	100%	0

Table 5. One Attack Dataset Results.

This phase is considered to be the easiest one since there is only one attack per normal. Having an even distribution of normal and one of the attack's records makes it straightforward for both neural networks to distinguish between the two. The goal of this phase is to show that the normal records could be told apart of the abnormal records, thus attacks. The success of this stage lets us conclude that each attack is indeed different in its pattern and activity from the normal behavior.

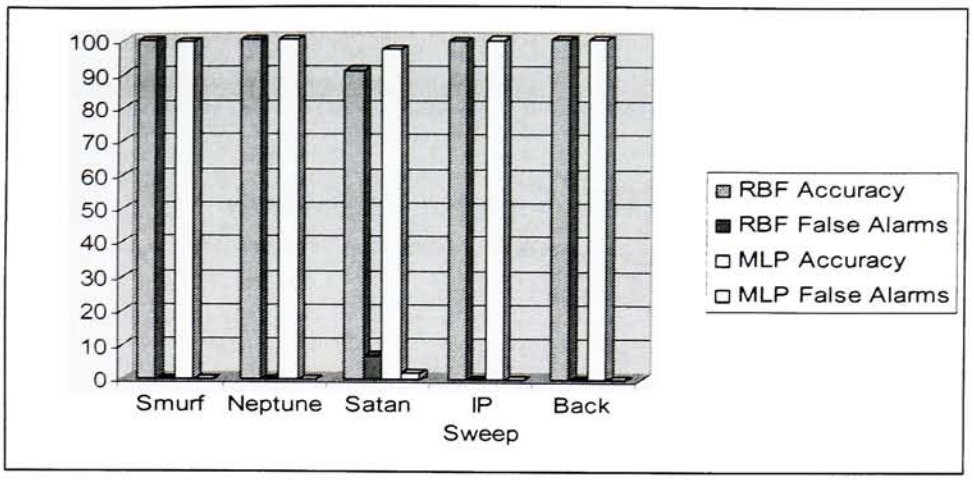

Figure 11. First Phase of the First Stage Graph.

For the second phase of the first stage of the experiments, five different attacks were used in the training set. Normal behavior records, as it was mentioned before, was considered as an attack, thus total of six attacks were used in this stage. In order to proceed to the next level of the experiments, attack classification, it was important to prove that the attacks are distinguishable not only one from another, as it was shown in the first stage, but also, one from the group of others. Therefore, six different experiments were held to prove this idea. 50% of the training set consisted of the concentrated attack, i.e. the attack that had to be differentiated from the others. Other 50% were evenly distributed between other attacks, i.e. 10% per attack. For example, normal behavior records needed to be defined. 50% of the training set for this assignment consisted of the records of normal behavior and other 50% contained records of Smurf, Neptune, Satan, IP Sweep, and Back attacks. All records were in random order. Table 6 and Figure 12 demonstrate the results of this experiment. As shown in the table, the accuracy for differentiating the attacks is quite high for both Neural Networks. The lowest accuracy is 91% for Satan and the

highest is 100% for Smurf, Neptune, and Back. These results let us make a conclusion that attacks can be differentiated, thus classified.

Attack Name	RBF Accuracy	RBF False Alarms	MLP Accuracy	MLP False Alarms
Smurf	100%	0	99.5%	0
Neptune	100%	0	100%	0
Satan	91%	7%	97.2%	2%
IP Sweep	99.5%	0	99.9%	0
Back	100%	0	100%	0
Normal	98.0%	1%	96.8%	2%

Table 6. Five Attack Dataset Results.

The main purpose of this stage was to show that neural networks are capable of differentiating between attacks, thus can single out an attack, or normal behavior pattern from a set of records. Due to the chosen approach, even distribution between the detecting attack and the group of the attacks, we managed to train the neural networks so that they would be able to successfully identify the trained attack. Both neural networks performed very well. They managed to detect attacks on a high level, showing 90% - 100% accuracy.

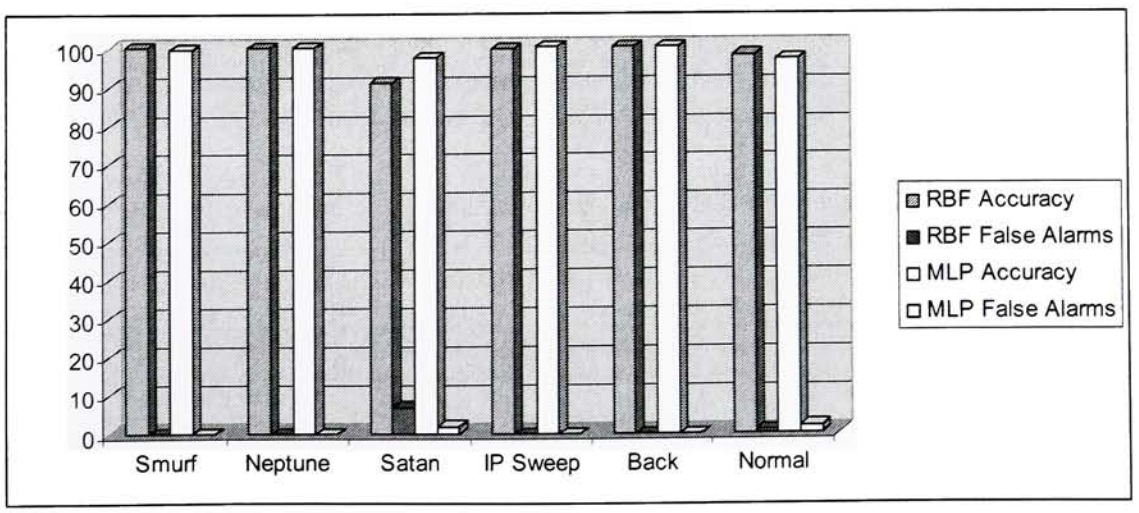

Figure 12. Second Phase of the First Stage Results.

Before going to the next stage, it was decided to enlarge this experiment and introduce new attacks to the training set. A neural network, which was trained to detect normal behavior, was chosen for this purpose. New attacks were added to the testing set: PortSweep and WarezClient. Neither of the neural networks was trained for these attacks. It was interesting to see their reaction to this introduction.

Radial Basis Neural Network successfully picked up both of the attacks, detecting 93.9% of the PortSweep attack, and 96% of the WarezClient attack. Multi Layer Perceptron Neural Network managed to detect 86.6% of PortSweep, and 96.1% of WarezClient. In both cases false alarm rate was under 5%.

In conclusion of the second stage, it is possible to state that we managed to create a neural network intrusion detection system that is capable of identifying attacks that were included into the training set as well as the attacks that were not included into the training set, thus new or unknown attacks. To take the experiments to the next level, we introduce the second stage.

Second Stage – Known Attacks Classification

For the second stage of the experiments Neural Networks with six outputs were used. At this level there was an attempt to create an Intrusion Detection System that is capable of classifying the attacks. A dataset of five attacks and normal behavior records were used. The attacks were evenly distributed in the dataset. Table 7 and Figure 13 demonstrate the results of this experiment. As we can see the accuracy of classifying attacks is 93.2% using RBF Neural Network and 92.2% using MLP Neural Network. The results were very close and the difference is statistically insignificant. In most cases the Networks managed to classify an attack correctly. The false alarm rate (false positive) is very low in both cases, missed attacks rate (false negative) is not high either, and the misidentified attacks rate (misclassification of the attacks) is 5%-6%. Overall, it is possible to conclude that both Neural Networks managed to accomplish the second stage of the experiments and were capable of classifying the attacks. Therefore, the environment for the third stage of the experiments was set.

	Accuracy	False Alarms	Missed Attacks	Misidentified Attacks
RBF	93.2%	0.8%	0.6%	5.4%
MLP	92.2%	0	2.1%	5.7%

Table 7. Attacks Classification Results.

After the success of the first stage, we show that neural networks not only capable of identifying attacks and normal behavior, but also are able to classify attacks. We are not introducing new, unknown attacks yet, leaving them for the third stage, but we observe the networks' behavior in dealing with multiple attacks and their classification. As it is shown, both attacks were successful in classifying trained attacks on a high level.

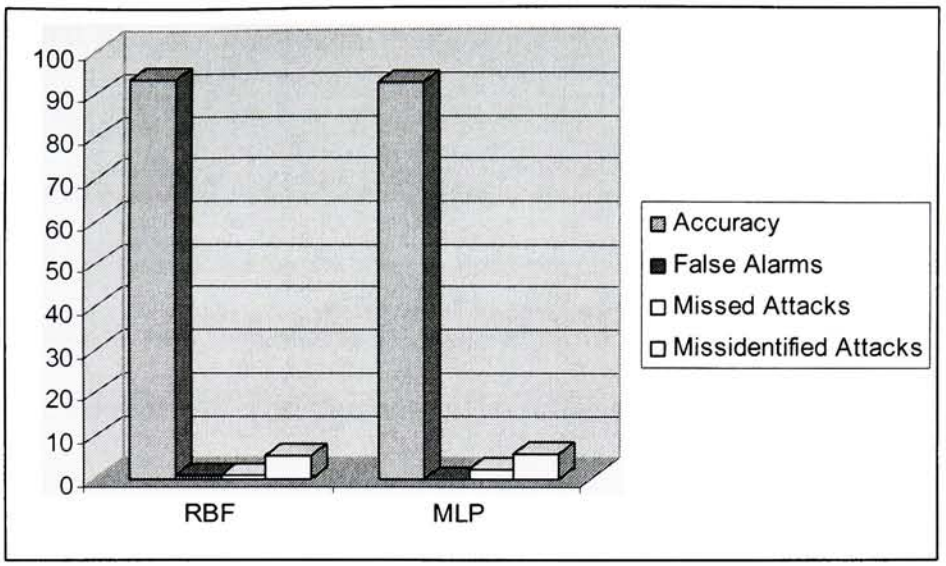

Figure 13. Second Stage Results.

Third Stage - Unknown Attacks Identification

For the third and the final stage of the experiments we used the trained Neural Networks from the second stage. The Networks were trained to classify the following attacks: Smurf, Neptune, Satan, IP Sweep, Back, and Normal behavior records. At this point we proceeded with the most interesting and exciting phase of the experiments – untrained (unknown) attack classification and identification. As it was mentioned earlier, five attacks were chosen to be used for this purpose: Buffer Overflow, Guess Password, NMap, Teardrop, and Warezclient. Datasets of these attacks were sent into the trained Neural Networks. Table 8 and Figure 14 demonstrate the results. As the table shows, RBF neural network managed to identify the unknown attacks as one of the trained attacks in most cases. As for the MLP Neural Network, it succeeded only with NMap and Guess Password attacks. In other cases it identified the attacks as normal behavior. Thus, RBF displayed more capabilities in identifying unknown attacks while MLP failed in some cases.

Attack Name	MLP	RBF
Buffer Overflow	53.3%	96.6%
Guess Password	96.2%	100%
NMap	99.5%	100%
Teardrop	1%	84.9%
Warezclient	8%	94.3%

Table 8. Unknown Attacks Classification Results.

At the final stage we introduced unknown attacks to the trained neural networks. It was interesting to see how these networks would react to this introduction. As we can see, Radial Basis Neural Network new attacks on a very high level, thus we can conclude that

the goal of this stage is fulfilled. At this point to go on with the analysis of the achieved results, and comparison them to the results of the previous researchers.

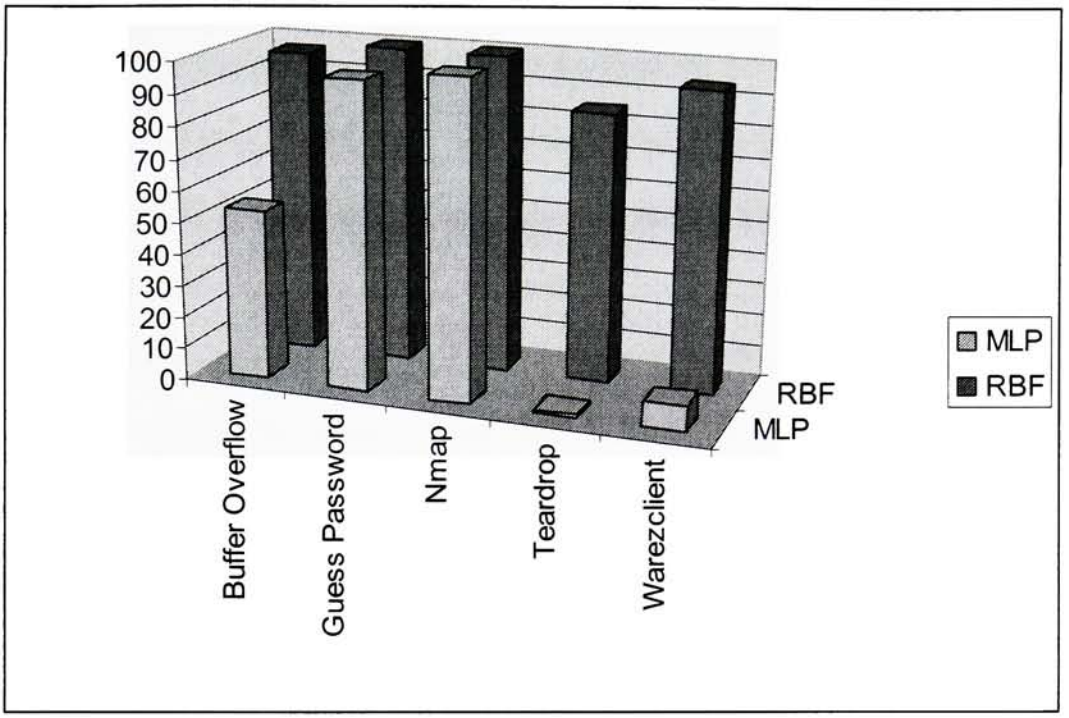

Figure 14. Third Stage Results.

Result Comparison

As the previous research indicates, there were many attempts to detect and classify attacks. The winner of the last KDD intrusion detection competition, Dr. Bernhard Pfahringer of the Austrian Research Institute for Artificial Intelligence, used C5 decision trees, the second-place performance was achieved by Itzhak Levin from LLSoft using Kernel Miner tool, and the third-place contestants, Vladimir Miheev, Alexei Vopilov, and Ivan Shabalin of the company MP13, used a decision tree based expert system [26]. The winner was determined based on the cost of the system, while the results were marginally close. Also, we note the results of the most recent research made by Maheshkumar Sabhnani and Gursel Serpen of the Ohio University who used a multi classify model to achieve even better results than the winner of the KDD Cups contest [37]. Table 9 and Figure 15 compare the mentioned above results. As we can see, in some cases accuracy of the classification is as low as 8.4%, which is totally not acceptable. The main problem with the approach they had chosen was that they used all attacks in the dataset, though many of those attacks did not have enough records for training, as we outlined after the data formatting and optimization took place. If an attack does not have enough presence (IMAP attack had only 12 records), it should not be used for training. Also, they grouped the attacks, what potentially can lead to a misdetection since not all of the attacks in the same group have identical signatures and patterns. Thus, a different approach was chosen to detect and classify attack. The main advantage of this approach was data formatting and the training dataset grouping, which allowed us to increase the accuracy rate up to 100% in some cases, and, the most important advantage, to achieve a high percentage of identification of the attacks that were not included into the training set.

		Probe	DoS	U2R	R2L
KDD Cup Winner	*Accuracy*	83.3%	97.1%	13.2%	8.4%
	False Alarms	0.6%	0.3%	0.1%	0.1%
KDD Cup RunnerUp	*Accuracy*	83.3%	97.1%	13.2%	8.4%
	False Alarms	0.6%	0.3%	0.1%	0.1%
Multi-Classifier	*Accuracy*	88.7%	97.3%	29.8%	9.6%
	False Alarms	0.4%	0.4%	0.4%	0.1%

Table 9. Result Comparison.

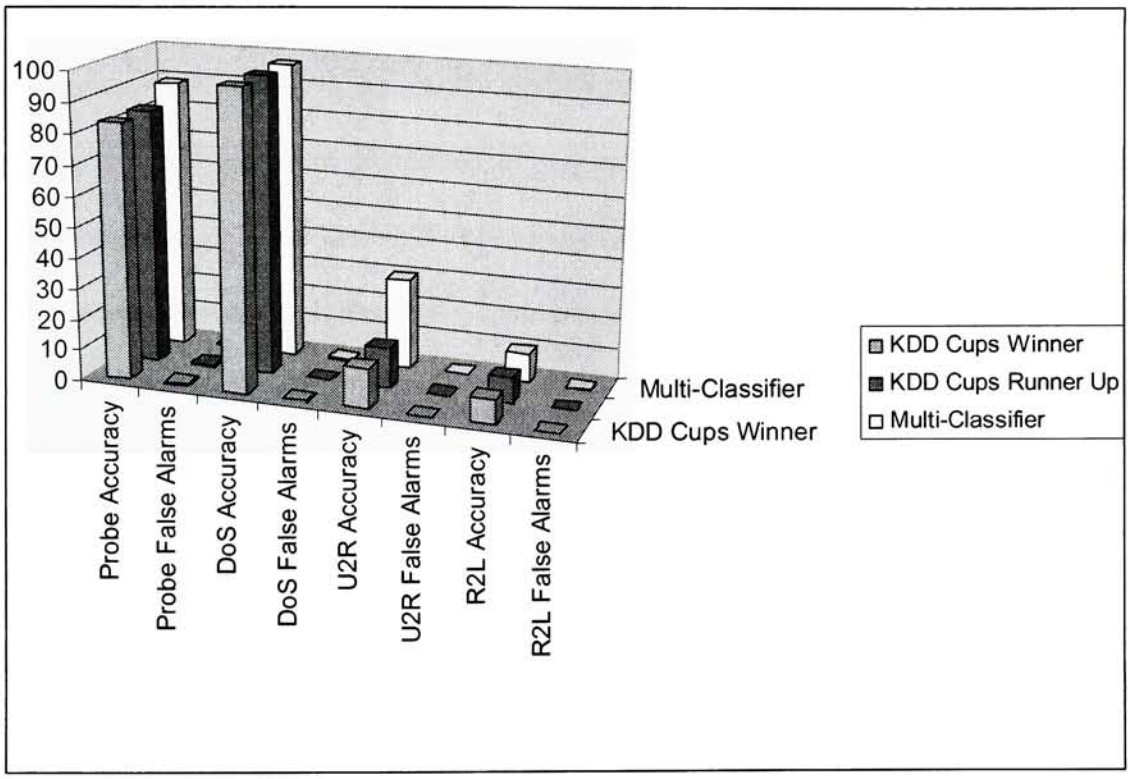

Figure 15. Result Comparison.

To make an objective comparison of the systems' results, it was decided to evaluate the average performance of each system. Table 10 and Figure16 demonstrate and graphically show the achieved comparison.

	Avg. Accuracy	Avg. False Alarms
KDD Cups Winner	50.5%	0.275%
KDD Cups Runner Up	50.5%	0.275%
Multi-Classifier	56.35%	0.325%
MLP IDS	63.34%	0.315%
RBF IDS	93.3%	0.422%

Table 10. Comparison of the Averages.

As it shown in the table, we took the average performance of each system. KDD Cups Winner, Runner UP, and Multi-Classifier Model was averaged based on their performance in classifying the attacks into 4 different groups. MLP and RBF IDSs were averaged based on their performance in classifying the attacks and detecting and classifying the unknown attack. As it is shown in the graph and the table, it is possible to see that the average performance of the systems is increasing significantly with each development. At the early stage, KDD Cups Competition, which was held back in the 1999-2000, the average performance of the system was 50.5%. The next generation of the IDS managed to increase its performance to 53.35%. And now, with our approach to data formatting, optimization, and training, it is possible to create a system that is capable of identifying attacks on acceptably high level of 93.3%.

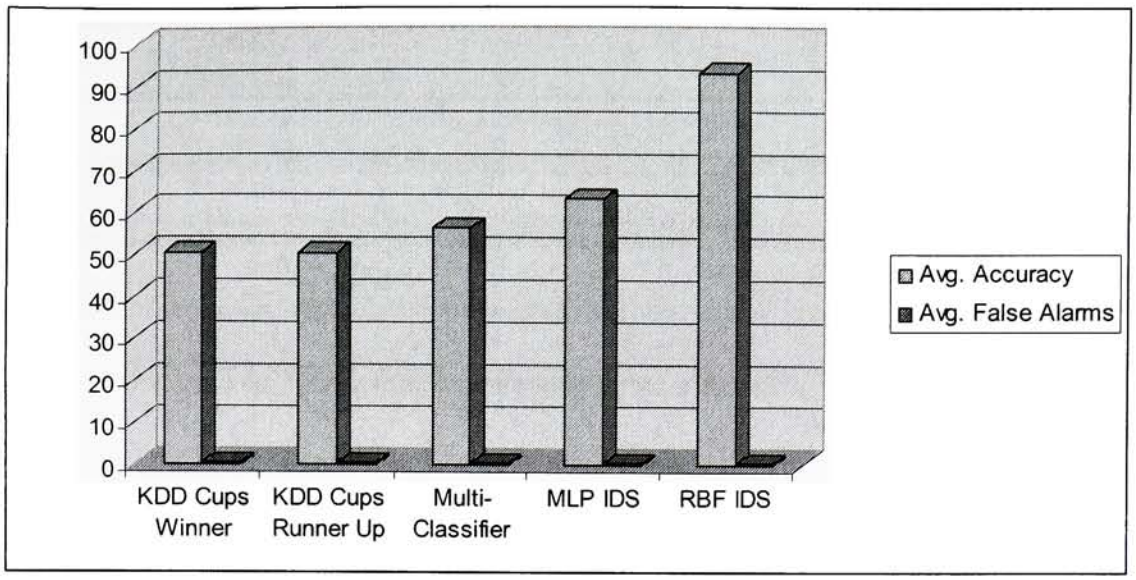

Figure 16. Comparison of the Averages.

To explain the better results we have to show the difference between our approach and the approach of other researchers. First, we have chosen a different strategy in preprocessing. Before using the dataset, we made a thorough analysis of the given data. We found out that there are a lot of repeated records. It was obvious that some attacks, such as Smurf were taking more than 50% of the whole dataset, and some attacks have only 10 or even less records. To optimize the dataset, to make it appropriate for the training and testing, we wrote a tool that was capable of resolving mentioned above problems and prepare the dataset for the neural networks to use. So, the dataset was optimized: repeated records were removed, dataset was split into multiple files, one attack per file. After the statistics were computed, we chose those attacks that had more representation in the dataset, thus the attacks with insignificant number of records were omitted.

The second very important difference was the training set composition. After the records were converted into the neural network readable form, i.e. all values were mapped and scaled into the range [0:1], we created training sets, trying to keep the even distribution of the attacks in the set. In other words, if it is important to identify a normal behavior from the set of records, the records of the normal behavior has to take 50% of the training set. Other 50% should be evenly distributed in the group of attacks.

Third, we chose to classify each attack, when others were trying to classify the attacks into different groups. Group classification could potentially lead to a confusion, since though the attack belong to the same group, i.e. trying to achieve the same goal, they have different signatures, patterns, and set of actions, thus could be misclassified.

Overall, due to neural networks optimization, removing attacks with insignificant number of records, removing repeated records, and taking a single attack approach instead of a group approach we managed to show better results than the results of other researchers.

6 Conclusion

As it was mentioned before, Computer Security remains one of the most researched fields in the area of Computer Science. Modern commercially used Intrusion Detection Systems employ the techniques of expert systems that require constant updates from the vendors. These updates make the IDS static, not flexible, and not capable of detecting new attacks without new batches. To improve the security, a lot of researchers put efforts to utilize Artificial Intelligence techniques in the area of Intrusion Detection, in order to create systems capable of detecting unknown attacks, or/and learning new patterns by themselves.

Benchmarks were created to standardize and compare the work of different investigators of this problem. Competitions were held to attract the attention of new researchers. In the most cases Neural Networks were used to detect attacks, and decision making trees were used to classify them.

The results and conclusions of previous researchers demonstrate the superiority of Decision Making Trees for attack classification, though in some cases even these trees were not able to perform in classifying the attacks on a high level.

After extensive study, we decided to come up with a unique solution, and approached the problem with a new dataset formatting and optimization technique. A library of attacks was created. This library was based on the benchmark provided by the MIT Lincoln Lab that was optimized by the KDD Cups. After the data was carefully formatted and

optimized, it was decided to use and compare two different Neural Networks in attack detection and classification. Neural Networks were chosen due to their abilities to learn and classify. Trained Neural Networks can make decisions quickly, making it possible to use them in real-time detection.

Both Networks managed to perform well on the known set of attacks, i.e. attacks that they were trained to identify and classify. After new attacks were added to the testing set, i.e. attacks that were not included into the training set, Radial Basis Function Neural Network performed significantly better than Multiple Layer Perceptron with the detection rate between 80% and 100%, and the false alarm rate not greater than 2%.

When we compared these results to the results of previous work, it was notable that the chosen technique had its advantages. First of all, we managed to correctly detect the attacks. Second, classification of the trained attacks was successful with the rate of 90-100%. Third, and the most important, we were able to detect new unknown attacks, which were not included into the training set. The accuracy of detecting new unknown attacks was between 80% and 100%.

After the work is done we conclude that with appropriate data formatting, optimization, and dataset composition, Neural Networks display a very good performance and potential in detecting and classifying trained attack, as well as new unknown attacks that were not included into the training set. Thus, the main goal of this research was accomplished.

Bibliography

1. Agarwal, R. and M. Joshi, *PNrule: A New Framework for Learning Classifier Models in Data Mining.* Technical Report TR 00-015, 2000.

2. Anderson, J., *An Introduction to Neural Networks.* MIT Press, 1995.

3. Aussem, A., A. Mahul, and R. Marie, *Queueing Network Modelling with Distributed Neural Networks for Service Quality Estimation in B-ISDN Networks.* Proceedings IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000.

4. Bace, R. and P. Mell, *NIST Special publication on Intrusion Detection Systems.* http://csrc.nist.gov/publications/drafts/idsdraft.pdf, 2004.

5. Bass, T., *Intrusion Detection Systems and Multisensor Data Fusion.* Communications of the ACM, 2000.

6. Batchelor, B.G., *Pattern Recognition: Ideas in Practice.* Plenum Press, 1978.

7. Bivens, A., et al., *Network-Based Intrusion Detection Using Neural Networks.* RPI, 2001.

8. C4.5, *http://www.rulequest.com.* 2004.

9. Cannady, J., *Artificial Neural Networks for Misuse Detection.* Proceedings, National Information Systems Security Conference on Neural Networks, 1998.

10. Cannady, J. and J. Mahaffey, *The application of artificial intelligence to misuse detection.* Proceedings of the 1st Recent Advances in Intrusion Detection (RAID) Conference, 1997.

11. Carpenter, G.A., S. Grossberg, and N. Markuzon, *Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps.* IEEE Transactions on Neural Networks, 1992. **vol. 3**.

12. Cunningham, R. and R. Lippmann, *Detecting Computer Attackers: recognizing patterns of malicious stealthy behavior.* MIT Lincoln Laboratory - Presentation to CERIAS, 2000.

13. Cunningham, R. and R. Lippmann, *Improving Intrusion Detection performance using Keyword selection and Neural Networks.* MIT Lincoln University, 1998(http://www.ll.mit.edu/IST/pubs.html).

14. DACS, *http://www.dacs.dtic.mil.* The Data and Analysis Center for Software, 2004.

15. DARAPA, *Intrusion Detection Evaluation.* MIT Lincoln Laboratory, 1998(http://www.ll.mit.edu/ist/ideval).

16. Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis.* Wiley, 1973.

17. Elkan, C., *Results of the KDD'99 Classifier Learning.* SIGKDD Explorations, 2000.

18. Ertoz, L., M. Steinbach, and V. Kumar, *Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data.* Technical Report, 2001.

19. Fox, K., R. Henning, and J. Reed, *A neural Network Approach Towards Intrusion Detection.* Proceedings of the 13th National Computer Security Conference, 1990.

20. Ghosh, A., A. Schwartzbard, and M. Shatz, *Learning Program Behavior Profiles for Intrusion Detection.* Proceedings First USENIX Workshop on Intrusion Detection and Network Monitoring, 1999.

21. Girardin, L. and D. Brodbeck, *A Visual Approach for Monitoring Logs.* 12th System Administration Conference (LISA '98), 1998: p. 299-308.

22. Ham, F.M., *Principles of Neurocomputing for Science and Engineering.* McGraw Hill, 1991.

23. Hartigan, J.A., *Clustering Algorithms.* John Wiley and Sons, 1975.

24. Hettich, S. and S.D. Bay, *The UCI KDD Archive.* University of California, Department of Information and Computer Science, 1999.

25. Kayacik, G., N. Zincir-Heywood, and M. Heywood, *On the Capability of an SOM based Intrusion Detection System.* IEEE 0-7803-7898-9, 2003.

26. KDD, *http://www-cse.ucsd.edu/users/elkan/clresults.html.* KDD Cups 99 - Intrusion Detection Contest, 1999.

27. Kohonen, T., *Self-Organizing Maps.* 1995.

28. Lee, W., S. Stolfo, and K. Mok, *Mining in a Data-Flow Environment: Eperience in Network Intrusion Detection.* In Proceedings of the 5th ACM SIGKDD, 1999.

29. Lee, W., S.J. Stolfo, and K.W. Mok, *A Data Mining Framework for Building Intrusion Detection Models.* IEEE Symposium on Security and Privacy, 1999.

30. Lee, Y., *Classifiers: Adaptive Modules in Pattern Recognition Systems.* Cambridge, MA: MIT, 1989.

31. Levin, I., *KDD-99 Classifier Learning Contest LLSoft's Results Overview.* SIGKDD Explorations, 2000. **vol. 1**.

32. Lippmann, R. and R. Cunningham, *Improving Intrusion Detection Performance Using Keyword Selection and Neural Network.* MIT Lincoln Laboratory, 2001.

33. LNKnet, *http://www.ll.mit.edu/IST/lnknet/index.html.* 2004.

34. Mukkamala, S., G. Janoski, and A. Sung, *Intrusion Detection Using Neural Networks and Support Vector Machine.* IEEE, 2002.

35. Planquart, J., *Application of Neural Networks to Intrusion Detection.* SANS Institute, 2001.

36. Rhodes, B., J. Mahaffey, and J. Cannady, *Multiple Self-Organizing Maps for Intrusion Detection.* GIT Information Technology and Telecommunications Laboratorys, 1999.

37. Sabhnani, M. and G. Serpen, *Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context.* EECS, University of Toledo, 2003.

38. StatSoft, *http://www.statsoftinc.com.* Electronic Statistics Textbook, 2004.

39. Werbos, P., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, 1974.

40. Wolfram, *http://www.documents.wolfram.com.* Wolfram Research Products, 2004.

41. Yampolskiy, R., L. Reznik, and D. Novikov, *Experimental Study of the Choice Between MLP and RBF Neural Networks for Character Recognition.* IEEE – WNYIPW, 2003.

42. Yeung, D.Y. and C. Chow, *Parzen-window Network Intrusion Detectors.* Sixteenth International Conference on Pattern Recognition, 2002.

43. Ypma, A. and R. Duin, *Novelty Detection using Self-Organizing Maps.* Progress in Connectionist-Based Information Systems, 1998. **vol. 2**.