NETWORK-BASED INTRUSION DETECTION USING NEURAL NETWORKS

RASHEDA SMITH
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
smithr2@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. 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 upon which it is attacking, 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 networks including the widely accepted Ethernet (IEEE 802.3) network use a shared medium for communication. Therefore, the packet sniffer only needs to be running on the same shared subnet as the monitored machines. Using this process of sniffing packets on a network, it is believed 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 or idea 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, is not a system that will prevent the attacks to a network, which is very hard to do, but is a system to alert that there is suspicious/anolomous activity occurring on a 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 are seen 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 where it creates communication between 2 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 machine will keep echoing to each other. With this, slowing of the network is easy and it would prevent 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 to a victim machine. The attack sends ICMP echo requests to many IP broadcast addresses. Every machine on the same networks as the broadcast addresses that are listening 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 different groups that have attempted and successfully use neural networks for intrusion detection each having different approaches. 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, 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. Testing the neural network, results were 17 out of 20 attacks were detected and they got 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 a two dimension space for visualization (Girardin, 1998). This process is similar to the project except they use the visual effects of self-organizing maps, where for this project, we want the self-organizing map to tell us which cluster the data belongs to not show us. Also, they would need someone to look at the graphical representation of the data and have the person determine if an attack is occurring. This project would not require such. The results 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 lets us know that it is not abnormal to use a tool to cluster network traffic together, if they contains patterns that are similar. This tool by aggregating clusters of network traffic they have found that it 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 encorporate 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.

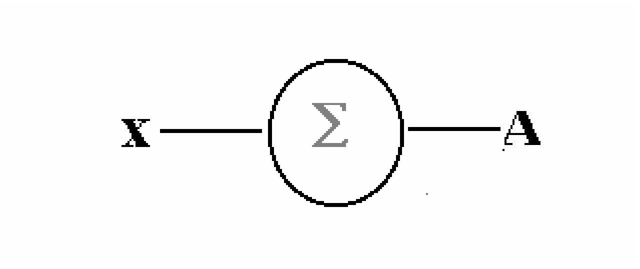


Figure 1 – A single neuron for a neural network taking input x with an output A

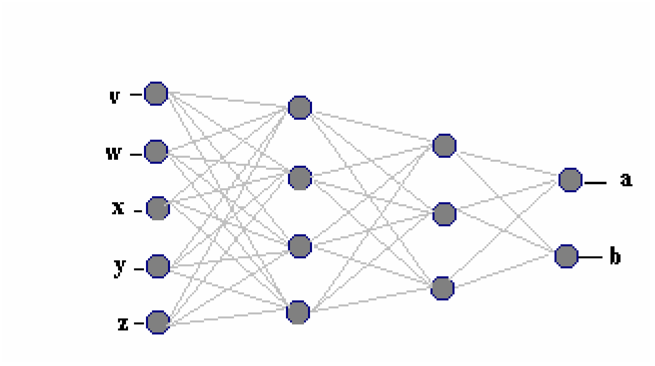


Figure 2 – Graphical representation of neurons interconnected receiving many inputs and produces 2 outputs

There are many different types of neural networks, which are not in the scope of this paper. However, the types of neural networks used for this project are 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 each take the input that is given to them and multiplies each input by a weight factor, takes the sum of the results of the multiplications and sends the results forward to the next set of neurons that are available. 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 by some error. To get the final result, each node in the network periodically adjusts the weights that are multiplied to the input until up to some error; resulting in values as close to the final result that was told to the neural network in the beginning stages as possible. After learning, then 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 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 infeasible in the number of inputs it can receive, while the self-organizing map is not due to the self-leaning and clustering structure of the map.

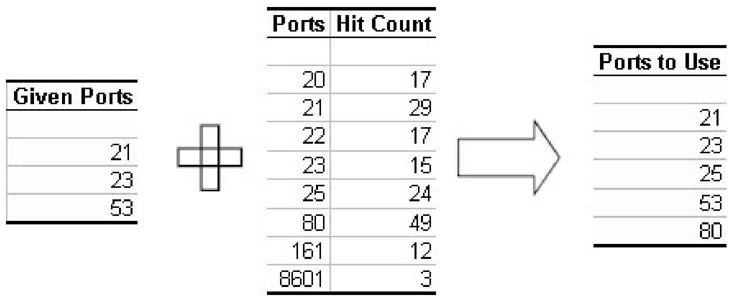
METHODOLOGY

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 is 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 is more knowledgeable. Professor Mark Embrechts developed the neural network software used for this project, a professor in the Decision Sciences and Engineering Sciences department. Having the access to the software allowed us to use a tool that supplies sufficient results than having to create a different model. 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 belive holds the information to signal 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 ☛*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 it is 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 beginning 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 ☛*t* to develop the length of our architectural learning phase. The network traffic is then observed for *F**☛*t* time, which the number of times source hosts access the many different ports on the target machine is cataloged. 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 is 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 which 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*.



Given Ports
21
23
53

Ports	Hit Count
20	17
21	29
22	17
23	15
25	24
80	49
161	12
8601	3

Ports to Use
21
23
25
53
80

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*☚t*, set A contains the ports and the number of hits the machine received during *F*☚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 from a different subnet. With this fact it was 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 ☚*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 ☚*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.

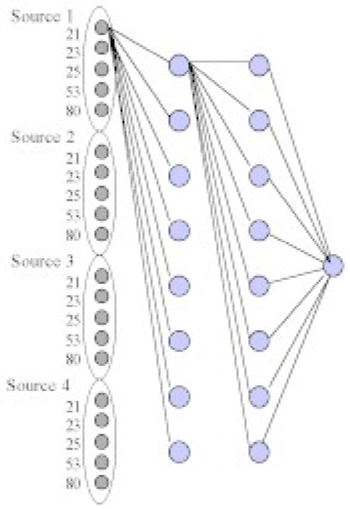


Figure 4 – Neural Network with 4 sources

Gathering total hits per source during ☚*t* creates a problem for the neural network. The number of sources received may change frequently from interval ☚*t* to interval ☚*t*, 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 ☚*t*, 3 sources made connections to destination x, creating an input size of 12 for the neural network. At another ☚*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 ☚*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 per ☚*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 ☚*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 the number of sources obtained per ☛*t*, the input to the neural network would remain fixed to N * M input nodes in which N is the number of cluster/classes of sources and M is the number of ports monitored on behalf of the target. 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 implementation of whether the data currently being looked at is an attack or not.

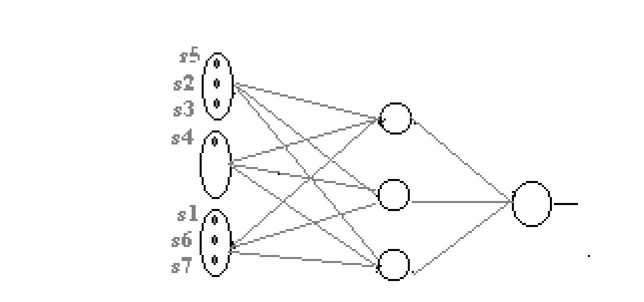


Figure 5 – Example of clusters of sources as input for the neural network

Figure 5 is an example of a graphical representation of the neural network containing 3 clusters of aggregated sources and their hits to the monitored ports, as input, one hidden layer and one output layer which outputs if the data seen has attributes of an attack or if it has attributes of normal traffic to the host. The self-organizing map is the tool that decides 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, PHASE I

Collecting Data files

In the beginning, in order to create the architectural learning phase and the introductory phase of the neural network learning phase, network traffic had to be gathered, which came from the DARPA Intrusion Detection evaluation program 1999 data set. The DARPA Intrusion Detection evaluation contained 5 weeks of sniffed network traffic. Of those, the data collected were the 4th and 5th weeks of sniffed network traffic. These two sets of data were selected since they both contained network-based attacks, normal network traffic, as well as documentation about the names of the attacks, start time of each attack, and the duration of each attack. This data is helpful, for the reason that in training the neural network, it needs to be given examples of both normal and abnormal behavior so that it can make decisions about the data it receives after training. If the neural network were trained on only normal data, it wouldn't be able to make decisions about an attack since the training data didn't contain any attacks. If the neural network were trained on only attacks, it would give flag everything it sees as an attack being biased on the data that it was trained which was on attacks. Based on those two facts, the neural network needs to be trained on an even number of both normal and abnormal traffic. Each data file obtained from the evaluation program consists of sniffed network traffic starting at 8:00am one day and ends 6:00am the next day. The files obtained are in binary format, where the only way to view the contents of the files are by running tcpdump (a network sniffing tool) on the contents of the file.

Gather.java

During this phase, work was being done in the projects/intrusion space on the machines in the department to ensure that there was enough space for work on this project. A problem with working in the intrusion space was that there were no access to the tcpdump program that will allow access to the contents of the sniffed network traffic. Therefore, it was created on a Linux machine (where unlimited access to tcpdump was available), which ran tcpdump as a process and opened the binary files and created a simplified parsed version of the file in text format. Tcpdump was run using the format: tcpdump –qnttr <filename>. The most important option used to run tcpdump is the tt option. This option prints an unformatted timestamp for each packet. Normal timestamp in tcpdump is written in the hh:mm:ss.microsecond format, while the unformatted tcpdump format is the number of seconds since 1970. Parsing of the data contained in the binary files by the program created a smaller of the file containing only the information of the packet pertaining to the project, which were the timestamp of the packet, who it came from, who it was destined to and the port that the packet was destined:

Original data in the binary file were in the format

```
922713037.440370 135.13.216.191.1559 > 172.16.114.148.21: tcp 0 [tos 0x10]
```

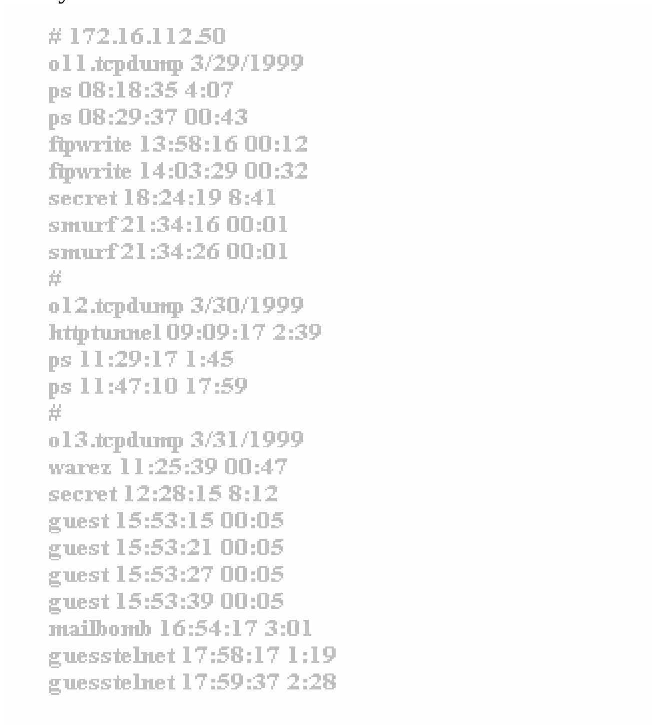
Parsed data in text file

```
922713037.440370 135.13.216.191 172.16.114.148 21
```

Each field in this new file is important to the collection of source data for the neural network. The timestamp is used to aid in the differentiation from attacks and normal traffic, since information about the attacks that are in the binary file were documented on the DARPA Intrusion Detection evaluation program and separating the traffic based on the timestamp of the packet would be the only option. Also, the timestamp of the packets are used to help manipulate the intervals for collection of the data for the neural network. The source is important, being one of the basis for the collection of hits to the ports, since the clustering techniques uses the information gathered per source to create correlations between the collected data. The destination is also important simply being, the machine that will be monitored by the system. Lastly, the port number is important in keeping track of the number of times there were hits to the ports by different sources during a specific time interval. The extra fields about each packet aren’t needed, given that they are not used for implementation of the project.

Configuration file

A configuration file called Neural.config was created that contains configuration information for the collection of data for the neural network. This configuration file contains all of the information about the attacks pertaining to the data file that the attacks occurred. The premise behind creating this configuration file is to be able to run the finished project once on a data structure that contains all the information needed for data collection. This configuration file supplied information that is used to help gather the hits to the ports for attacks that is given to the neural network, such as the binary files to use, the name of the attack, the start time of the attack, and the duration of the attack. For each file, the date of the binary file is needed to help convert the start times of the attacks to the same unformatted time stamp used when tcpdump was run to create the parsed data file. This file can contain as many information about the binary files as needed:



```
# 172.16.112.50
o11.tcpdump 3/29/1999
ps 08:18:35 4:07
ps 08:29:37 00:43
ftpwrite 13:58:16 00:12
ftpwrite 14:03:29 00:32
secret 18:24:19 8:41
smurf 21:34:16 00:01
smurf 21:34:26 00:01
#
o12.tcpdump 3/30/1999
httptunnel 09:09:17 2:39
ps 11:29:17 1:45
ps 11:47:10 17:59
#
o13.tcpdump 3/31/1999
warez 11:25:39 00:47
secret 12:28:15 8:12
guest 15:53:15 00:05
guest 15:53:21 00:05
guest 15:53:27 00:05
guest 15:53:39 00:05
mailbomb 16:54:17 3:01
guesstelnet 17:58:17 1:19
guesstelnet 17:59:37 2:28
```

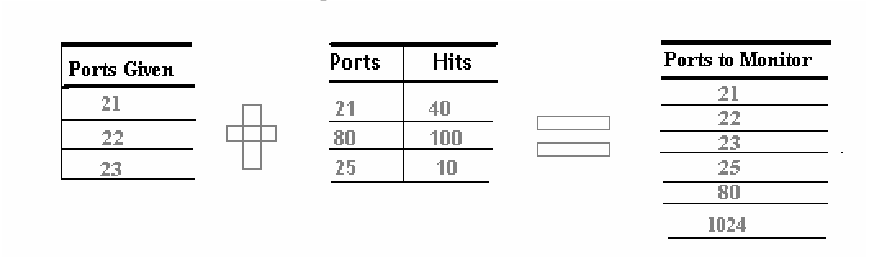
Figure 6 – Example of configuration file

GetConf.java

GetConf is used to read the configuration file and set up the data structures to start the collection of source data for the neural network. The program can be run by:

javac GetConf <configuration file><host you want to monitor><port 1>…<port n><number of extra ports to get><interval for architectural learning><interval ☛t>.

- **Configuration file** - The program reads the attacks from the specified configuration file. By looking at figure 6, it can be seen that there is a delimiter # between each file. The delimiter is used for the to help in the collection of sets of data. The data between the delimiter is read first, put into a data structure for accessibility, and then sent to the data collection process. On return, the process is repeated for every attack instance that is recorded in the file.
- **Host you want to monitor** – This is the IP address of the machine that the number of hits per source is collected for.
- **Port 1…Port n** – This is the set *KP* that the administrator chooses.
- **Number of extra ports to get** – The architectural learning phase uses this number to obtain the *FinalSet* list that was explained above. Before the *FinalSet* list is created, a check is made to see if at least extra number of ports active at the host. If the total has not met the minimum needed, then the remainder needed is randomly chosen to fulfill the amount of ports to be monitored for the host.



Ports Given
21
22
23

Ports	Hits
21	40
80	100
25	10

Ports to Monitor
21
22
23
25
80
1024

Figure 7

For the figure above, the administrator entered 3 ports (shown in the Ports Given table), and is requesting 3 extra ports. At the end of the architectural learning interval, only 3 ports were found that were active. The results for ports to monitor will (shown in Ports to Monitor) the ports given by the administrator, along with the 2 ports that are not from the set of given ports and a random port number. Altogether, there will be 6 ports that will be monitored at the host machine.

- **Interval for architectural learning** – This parameter sets the length of the architectural learning phase. At the end of this phase, the *FinalSet* list is formed.
- **Interval ☛t –** The collection of hits per source is collected ever ☛*t* time interval. At the end of each ☛*t* the results collected are put into a data structure for later use. During this interval, the number of sources varies from interval to interval, varying from 0 to N. There can be no sources seen in certain time intervals for two reasons, one being the fact that no source might be communicating to the host at that time, and the

other being that there is communication between a source and the host, but not to any of the ports in the list of ports that are being monitored. Those connections to the ports not being monitored cannot be added to the list since adding them will change the architecture to the neural network, requiring new training of the network, as explained above, the input to the neural network is fixed and changing the input will render it useless, unless the architecture is changed.

As the data is gathered per source according to the ports that are being monitored, at the end of each interval, the data is placed into a data structure for later use. The data collected containing normal packet behavior is placed into a separate data structure than the data collected containing attack packets. These two data structures will be used to distinguish the attack data from the normal data, when they are sent to the neural network for training.

Parse.java

The backbone to the collection of the data for clustering and input to the neural network is the file. In the beginning the *FinalSet* is found during the architectural learning phase for the architectural learning phase time interval. After this phase has ended, the collection of normal and attack data structures can be formed. For normal data, source data is collected every ☚*t* interval. At the end of each ☚*t*, the data is placed into the normal hits data structure. Attack data is collected a little differently than the normal data, since attack data needs the start of the attack and the duration of the attack to help in the collection of attack information. To collect the hits to the monitored ports per source in an attack, if the time timestamp for the incoming packet matches the time for which an attack starts, the hits per source is gathered every ☚t interval during the length of the attack. This process guarantees that some normal traffic will be aggregated together with the traffic of an attack because if the interval for normal traffic has not ended before the attack started, then in order to keep the integrity of the algorithm, using a consistent time interval, the normal data needs to be merged with the attack data. If an attack ends before ☚t time period ends, then the remaining time is collected and saved as an attack. The gathering for the attack does not end if the attack ends before ☚t ends due to the fact the data is collected every ☚t time interval and the integrity of the data collected during this process needs to be consistent throughout the whole application.

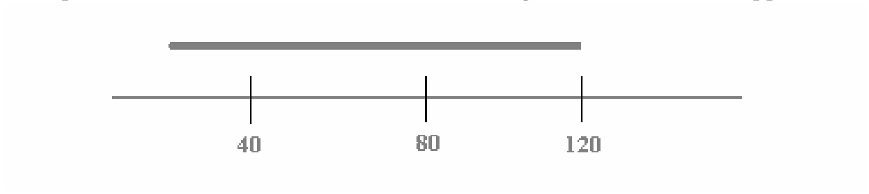


Figure 8

Figure 8 is an example showing that the attack started during the 1st ☚t where normal data was being collected, and ended at the end of the last interval.

Before the attack started the data gathered is normal, but since the attack started during the time the normal traffic was gathered, the group of sources will now not be seen as normal data but be seen an attack data. If not, it will cause inconsistencies in the data to train the neural network, where each data per source will be of different intervals. To prevent this, the data is transferred to be part of an attack, meaning, the data collected during the attack in this figure is during a total of 3 ☚ts instead of 2 or 2.5. The next figure shows a similar example as above with the difference of the attack starting at the beginning of ☚t and ends before another ☚t ends; but the same premise stays with there being 3 intervals of the attack.

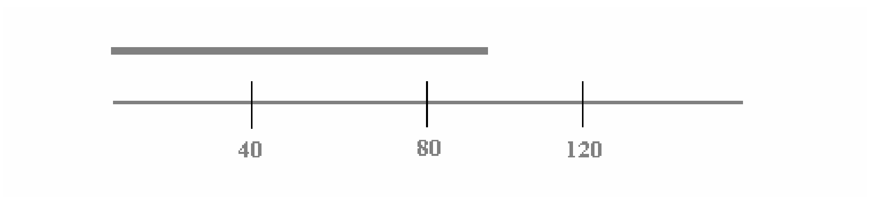


Figure 9

ConvertTime.java

In order to use the time given for the start of attacks, they need to be converted to the same format as the unformatted timestamp from above. In order to do so, the date obtained from the configuration file is needed, and is merged with the time of the attack. The final result is the unformatted time for the attack, which can be used to determine if an attack is occurring.

CONTTRIBUTIONS MADE TO THE PROJECT - PHASE II

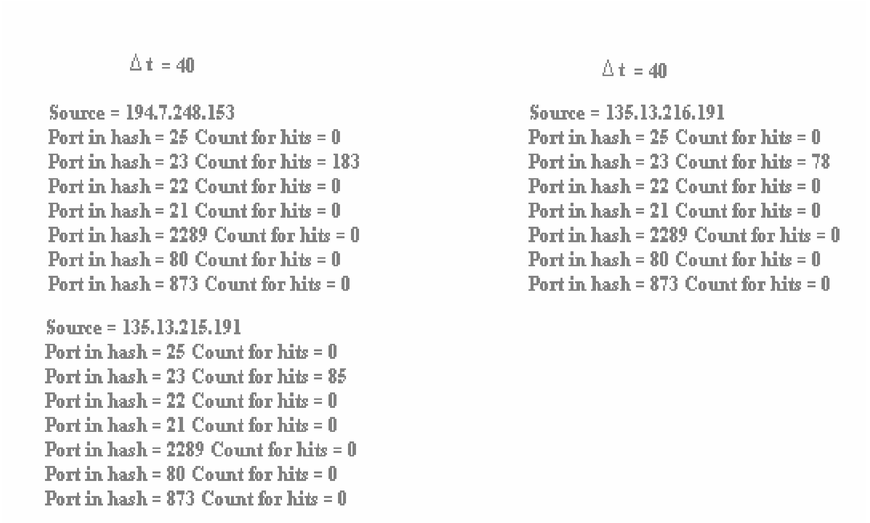
The programs created in Phase I is used in Phase II, with a few minor changes. The first change is that Gather.java is no longer needed. This is because the project has been moved to a set of machines that allow access to the tcpdump program, which will enable direct use of the binary files without having to convert it to another format before it can be used. Not doing that process decreases the run time of the project where doesn't have the overhead of creating and reading from multiple files. With the removal of Gather.java, GetConf.java has been changed to run tcpdump using the same flags of –qnttr , on the binary files listed in the configuration file, and use the stream outputted from tcpdump, to read the packets from the network data. This way, it would be easy to use this system on a network, for the fact that it is configured to run tcpdump as a process allowing the collection of real-time network data. The second change is also added to GetConf.java, to combine the data structures containing attacks and normal traffic, and convert that structure to a binary file. This is done, so that if the same data is needs to be accessed multiple time, it should not be required to run everything from. After the data is collected, it is sent for clustering to create the inputs for the neural network.

STEPS TO OBTAINING RESULTS

Alan Bivens is wrote 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 for where the data belongs, the aggregation of the sources as explained in the introduction is done, and then the aggregation is sent to the neural network, for training. After training, the neural network will be tested to see if it can determine if the network traffic or not, by sending it the clustered data of new instances that it has never seen before.

CURRENT RESULTS

After all of the information is gathered for attacks and non attacks and put into their prospective data structures, the results about the source information received is shown below:



```
        ∆t = 40                                      ∆t = 40

Source = 194.7.248.153                          Source = 135.13.216.191
Port in hash = 25 Count for hits = 0            Port in hash = 25 Count for hits = 0
Port in hash = 23 Count for hits = 183          Port in hash = 23 Count for hits = 78
Port in hash = 22 Count for hits = 0            Port in hash = 22 Count for hits = 0
Port in hash = 21 Count for hits = 0            Port in hash = 21 Count for hits = 0
Port in hash = 2289 Count for hits = 0          Port in hash = 2289 Count for hits = 0
Port in hash = 80 Count for hits = 0            Port in hash = 80 Count for hits = 0
Port in hash = 873 Count for hits = 0           Port in hash = 873 Count for hits = 0

Source = 135.13.215.191
Port in hash = 25 Count for hits = 0
Port in hash = 23 Count for hits = 85
Port in hash = 22 Count for hits = 0
Port in hash = 21 Count for hits = 0
Port in hash = 2289 Count for hits = 0
Port in hash = 80 Count for hits = 0
Port in hash = 873 Count for hits = 0
```

Figure 10 – On the left for time interval 40 seconds there were 2 sources captured. On the right for another interval of 40 seconds, there was only 1 source captured.

This data containing the hits to the ports are the data that clusters the sources together for each interval. On input for clustering, the self-organizing map is given a total of 50 normal traffic and 50 abnormal traffic on data similar to figure 10. Based on the data it received, it formed 3 clusters. The self-organizing map then decides to which cluster does the sources in each interval belong. For example: from the above figure, and put both sources into cluster 2. Since there are 3 clusters and both belong to the same cluster, their data is aggregated together and sent to the neural network. At the neural network, input 1 which represents cluster 1 will contain values of 0s, input 2 which represents cluster 2 will contain the aggregated values from the sources, and input 3 which represents cluster 3 will contain values of 0s. After enough data is sent to the neural network, it can then be trained and tested to see if it can distinguish abnormal data from normal data.

FUTURE WORK

The only job that needs to be accomplished is the training and testing of the neural network to see what decisions it makes about the problem of distinguishing normal traffic from abnormal traffic. If it is discovered that it cannot make any correct decisions, then the actual structure of the neural network might probably need to be changed or the information that is being collected for input to the neural network might need to change. If so more work will be done in trying to perfect a system that uses a neural network for intrusion detection.

ACKNOWLEDGEMENTS

I would like to acknowledge Alan Bivens for his work on this project by helping to supply 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:

Chandrika Palagiri for creating the neural network expert that will detect the anomalies

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

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

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

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

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

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

Oliver Niggemann, Benno Stein, Jens Tölle. Visualization of Traffic Structures. IEEE International Conference on Communications, 2001. ICC 2001, Volume: 5 , 2001 Page(s): 1516 -1521 vol.5.