NSOM: A Real-Time Network-Based Intrusion Detection System Using Self-Organizing Maps

Khaled Labib and Rao Vemuri

Department of Applied Science
University of California, Davis
Davis, California, U.S.A.

Abstract. In this paper we describe an implementation of a network based Intrusion Detection System (IDS) using Self-Organizing Maps (SOM). The system uses a structured SOM to classify real-time Ethernet network data. A graphical tool continuously displays the clustered data to reflect network activities. Different system parameters such as data collection, data preprocessing and classifier structure are discussed. The systems shows promise in its ability to classify regular v.s. irregular and possibly intrusive network traffic for a given host.

I. Introduction

With the growing rate of interconnections among computer systems, network security is becoming a major challenge. In order to meet this challenge, Intrusion Detection Systems (IDS) are being designed to protect the availability, confidentiality and integrity of critical networked information systems. They protect computer networks against denial-of-service (DoS) attacks, unauthorized disclosure of information and the modification or destruction of data. The automated detection and immediate reporting of intrusion events is required in order to provide a timely response to attacks [1].

Early in the research into IDS, two major principles known as anomaly detection and signature detection were arrived at, the former relying on flagging all behavior that is abnormal for an entity, the later flagging behavior that is close to some previously defined pattern signature of a known intrusion [2]. NSOM, our Network-based detector using SOM, could be classified as an anomaly detection system. Anomaly detection attempts to quantify the usual or acceptable behavior and flags other irregular behavior as potentially intrusive [3].

We created a prototype system, NSOM, to classify network traffic in real-time. The system is implemented is a combination of C and TCL/TK. We continually collect network data from a network port, preprocess that data and select the features suitable for classification. We then start the classification process - a chunk of packets at a time - and then send the resulting classification to a graphical tool that portrays the activities that are taking place on the network port dynamically as we receive more packets.

Our hypothesis is that routine traffic that represents normal behavior would be clustered around one or more cluster centers and any irregular traffic representing abnormal and possibly suspicious behavior would be clustered outside of the normal clustering.

The remainder of the paper is organized as follows. Section II discusses other related work in the field. Section III discusses the problem that we are trying to solve. Section IV describes in detail the process of data collection and preprocessing. Section V describes the SOM structure used. Section VI presents the results obtained from the experiment and Section VII is the conclusion.

II. Related Work

Most of the related work in anomaly detection using Self-Learning utilizes ANN (Artificial Neural Networks) as in HyperView [4]. The system's normal traffic is fed to an ANN, which subsequently learns the pattern of normal traffic. The new traffic, including possible attacks, is then applied to the ANN and the output is used to form the intrusion detection decision.

Other systems utilize descriptive statistics by collecting uni-modal statistics from certain system parameters into a profile, and construct a distance vector for the observed traffic and the profile. If the distance is great enough the system raises the alarm. Examples of these systems are NIDES[5], EMERALD[6] and Haystack[7].

A system developed by [8], uses multiple self-organizing maps for intrusion detection. They use a collection of more specialized maps to process network traffic for each layered protocol separately. They suggest that each neural network become a kind of specialist, trained to recognize the normal activity of a single protocol.

Another approach that differs from anomaly detection and misuse detection considers human factors to support the exploration of network traffic [9]. They use self-organizing maps to project the network events on a space appropriate for visualization, and achieve their exploration using a map metaphor.

Both the last two systems use static logs and do not address the real-time issues that we address in the design of NSOM. We believe that real-time performance can only be achieved by minimizing the processing of data, and therefore using simpler designs. They also do not describe how to handle the problem of representing time in their work. We believe that time representation is an important element when considering network traffic considering that attacks takes place using a number of successive packets that are targeted towards a host in a finite time limit.

III. Why SOM ?

Unsupervised learning using SOM provide a simple and efficient way of classifying data sets. To process real-time data for classification we believe that SOM are best suited due to their high speed and fast conversion rates as compared with other learning techniques. Also SOMs preserve topological mappings between representations, a feature which is desired when classifying normal v.s. intrusive behavior for network data. That is, the relationships between senders, receivers and the protocols used amongst them, which are the primary features that we use, are preserved by the mapping.

Figure 1 depicts a block level diagram of NSOM. The diagram shows the different steps the system performs to achieve the real time classification of network traffic.

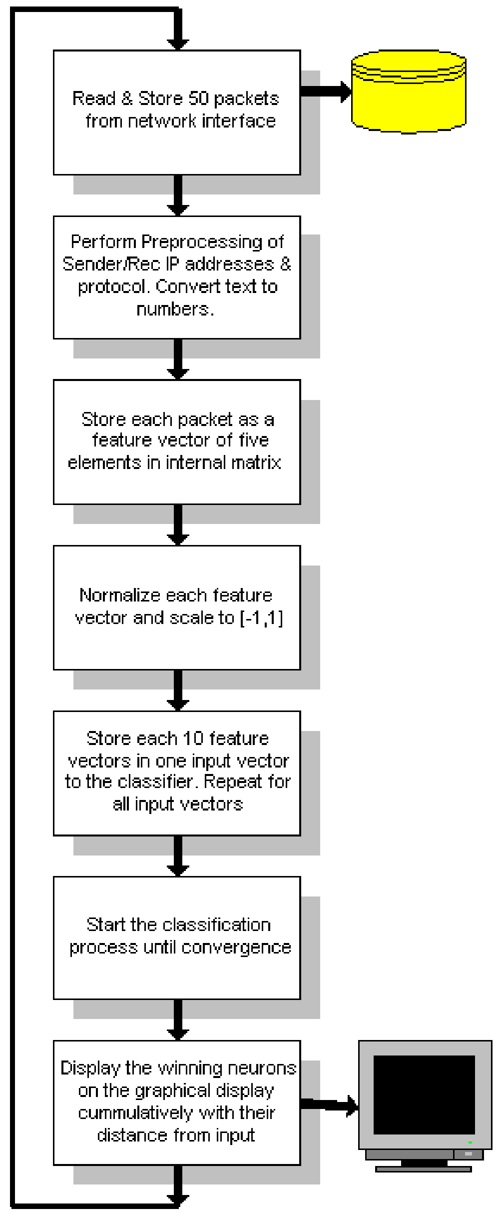

Figure 1: Block Diagram of NSOM

IV. Data Collection and Preprocessing

We used a host PC running Linux as our primary test bed. This system is connected to a network using an Ethernet controller. The subnet that the host is connected to has tens of other hosts, which are running several daemons such that data and control frames are constantly flowing across the subnet. We used a popular Linux tool called "tcpdump" for the purpose of data collection and filtering. Tcpdump is a powerful tool that allows us to put the Ethernet controller in a promiscuous mode to monitor all packet activities on the subnet. We can also use its powerful filtering capabilities to filter out unwanted traffic and isolate broadcast, multicast and control frames.

We used tcpdump to filter and collect all network traffic to or from our host, discarding packets that are intended for other hosts. Tcpdump is run as a background process, which dumps the information it collects into a file on a regular basis. Every time we collect 50 packets we store them in a file for further preprocessing and classification as described below, and then repeat the process. All the different system parameters such as the number of packets to collect per processing cycle and all the parameters associated with the classifier, are easily configurable in the source code to be customized for any given host. We used 50 packets here since this was the most suitable number to use given the amount of traffic involving our host system in our subnet. This value constitutes the "window" that we analyze packets through. If this value is too small, then there is a potential risk of losing important relationships between the packets that would otherwise show specific important patterns characteristics. If the value is too large, then the real-time effect could be lessened due to the fact that the graphical updates would be less frequent, especially for hosts with light traffic.

When writing the packets information to the disk in each processing cycle, we minimize the information written using special tcpdump flags and filtering commands. After receiving the packet information, we had to make a choice over which information from this file to use for the SOM classifier. The choice of which traffic features to represent and how to translate them in a form suitable for the SOM, will unavoidably involve highlighting certain aspects of the network activity while making other obscure or even invisible to the classifier [8]. We selected only a portion of the information received to serve as a feature list for the packet, as follows:

```
For each Ethernet packet received{
   - Extract the IP address of the destination: Use
the least significant two numbers only for
classification
   - Extract the IP address of the source: Use the
least significant two numbers only for
classification
   - Extract the protocol type
}
```

The IP addresses for both the destination and source are in the form of 4 decimal numbers separated by dots. (e.g. 192.138.45.3) We only select the least significant two numbers of these to represent the source and destination, separately, instead of using the whole numbers. Since the upper two numbers do not change frequently in our subnet, we decided not to use them since they could potentially pollute the classification results as being redundant background. NSOM could be changed to allow including the entire IP address if this behavior is desired.

Another important feature that we keep in the process of representing a packet is the protocol type. Protocol type can include and TCP/IP or UDP. All the different variations such as ICMP, ARP and RARP are supported. Since all protocol type names are decoded by tcpdump as text, we convert the text to a decimal number by adding the ASCII values of all its characters and we use at most 5 characters from each protocol type name for the representation. In our opinion, this provides a uniform representation of the protocol type.

Data normalization and scaling

A feature vector representing a packet consists of five features representing partial destination and source addresses and the protocol type. That is, two numbers for sender, two for receiver and one for the protocol type. Due to the large variations of these numbers we normalized each vector such that it components are in the range of [0,1]. This makes it more suitable for SOM applications. We used the standard normalization given by:

$$nv[i] = \frac{v[i]}{\sqrt{\sum_{K} v[k]^2}}$$

Where *nv[i]* is the normalized value of feature *(i)*, *v[i]* is the feature value of *i*, and *K* is the number of features in a vector.

During initial testing we found that the normalization of the feature vector was not providing acceptable classification results. So we further scaled the vector values to the range to [-1, 1]. This provided for better performance of the SOM classifier.

Time Representation

Even though packet arrival and departure times were explicitly available before the data was preprocessed, we decided not to use explicit time representation for reasons discussed in [1]. We rather used an implicit time representation scheme. In this scheme, *n* successive packet features are gathered to form one input vector for the classifier. We call this vector the SOM Input Vector. So the classifier looks at *n* packets at once in the same order they arrived at the network port. The value we chose for *n* in our experiment was 10. Again, NSOM can be configured for different numbers if so desired.

V. SOM Structure

We experimented with two SOM structures: Linear and Diamond structures. Diamond structure gave better classification results. For Linear structure, we updated the winning neuron along with a neighborhood distance of R, representing the number of neighbor neurons to update. In this case we chose a number of R = 1. For Diamond structure we updated the winning neuron along with a neighborhood distance of R. The neighboring neurons in this case were the top, bottom, left and right neurons of the winning neuron, which resembles a Diamond-like structure. In this case we chose R = 1, which means that four neurons would be updated in addition to the winning neuron, given a central neuron.

In our experiment there were 25 output neurons. In the case of the Diamond structure they are virtually arranged in a 5x5-matrix plane.

The SOM implementation we chose was a Kohonan Net with the winning neuron representing the one with the shortest distance as related to the input vector. The starting value we chose for $\eta = 0.6$. This value decrements by 0.5 in every epoch.

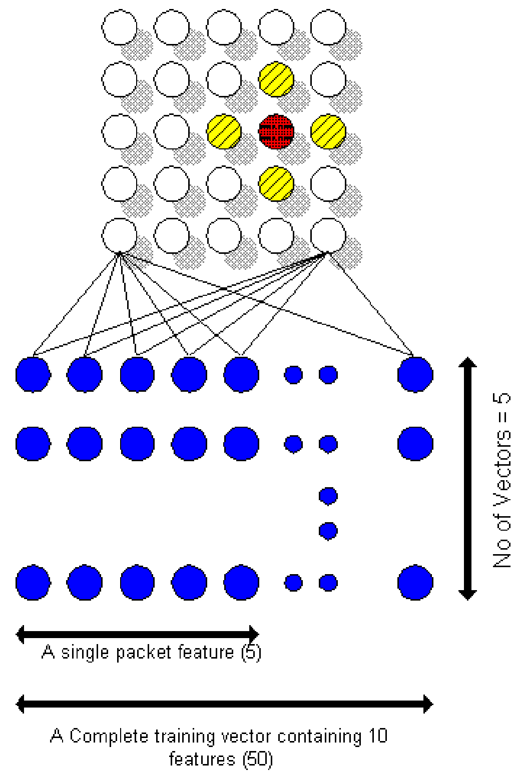

Figure 2: SOM structure and training vectors layout

After *m* successive training vectors are collected, normalized and scaled, the process of classification is started until we reach convergence. In our experiment we chose m = 5. When conversion is reached, meaning that no further changes are taking place in the winning neurons between successive epochs, the winning neuron values and their locations are sent to a graphical tool that displays these values in a two-dimensional form. The display maintains the old values as well to show the clustering and accumulation effects. During this time we start storing packets again from the network interface into a file as the following batch. On heavy network loads, we could practically drop few packets that would go by without reaching the classifier, but we believe that their would not be much risk involved with this situation, since it is difficult for an attacker to finish an attack with very few packets involved.

VI. Results

To test NSOM, we first obtained sample results statically by collecting different sample network traffic representing normal as well as DoS attacks. We looked at the output of the classifier for each case and noticed that all normal network traffic was clustered roughly between neurons 5 and 16. When we subjected the classifier to various simulated DoS attacks, such as frequent SYN packets and heavy ping (ICMP req) packets, we noticed that neuron activities began to be scattered much outside the normal cluster window indicated in Figures 1 and 2. The new range for activated neurons was expanded to cover between neurons 0 to 18 indicating a possible attack.

When we were more confident about the results, we tested NSOM in real-time. Network data were collected, classified and graphically displayed continuously in real-time. Similar behavior as with static testing was noticed.

It is interesting to note that the Y values, on the graph, of the attack neurons were much higher than those for normal ones. Since the Y values represent the distance of the winning neurons with respect to the input vector, we can conclude that these high Y value neurons represent uncommon and irregular behavior and therefore a possible attack.

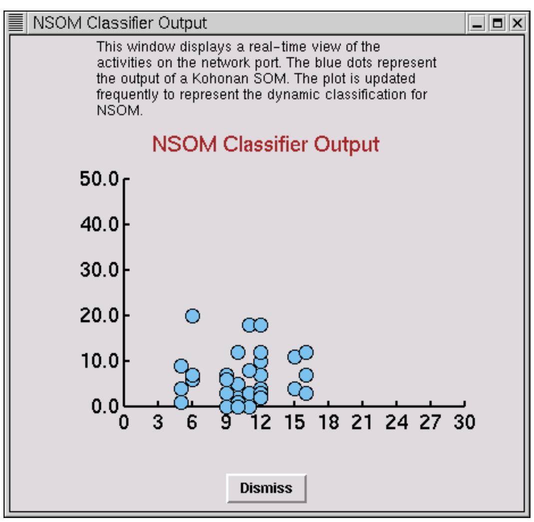

Figure 3: Output of classifier for normal traffic

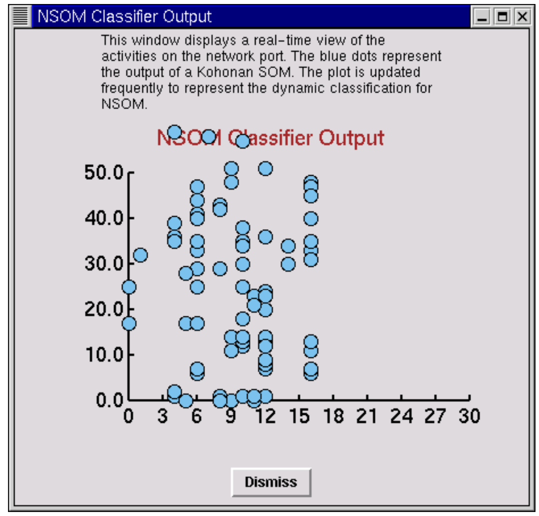

Figure 4: Output of classifier for a simulated DoS attack

The results support our initial hypothesis that similar network traffic that takes place routinely, that is from/to common IP addresses and common protocol type patterns could be classified by a close set of relatively fixed neurons. Thereby, abnormal behavior that could be a result of a DoS attack will be characterized by a different set of neurons that span a larger area on the output neuron map.

VII. Conclusion

We described the implementation of a prototype system for classifying real-time network traffic using Self-Organizing Maps (SOM) for the purpose of intrusion detection. We presented the motives behind using unsupervised learning for this purpose, our data collection and preprocessing procedures, how we represented time and our technique for displaying the classification results. We discussed the structure of our SOM and how we conducted the testing. The results showed that we were able to classify simulated DoS network attacks graphically as opposed to normal traffic by showing that the clustering of neurons was very different between the two.

References

[1] Lichodzijewski P., Zincir-Heywood A., Heywood M., "Host-based Intrusion Detection Using Self-Organizing Maps"

[2] Axelsson S., "Intrusion Detection Systems: A Survey and Taxonomy". Technical report 99-15,

Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, March 2000.
[3] Lane T., Brodley C., "An Application of Machine Learning to Anomaly Detection". National Information Systems Security Conference, Baltimore M.D. 1997.
[4] Debar H., Becker M., Siboni D., "A Neural Network Component for an Intrusion Detection System". Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA May 1992
[5] Anderson D., Frivold T., Valdes A., "Next-Generation Intrusion Detection Expert System (NIDES). Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park CA , May 1995
[6] Porras P., Neumann P., "EMERALD: Event monitoring enabling responses to anomalous live disturbances". Proceedings of the 20th National Information Systems Security Conference, Baltimore, Maryland 1997
[7] Smaha S., "Haystack: An Intrusion Detection System" Proceedings of the IEEE forth Aerospace Computer Security Applications Conference, Orlando, Florida, 1988
[8] Rhodes B., Mahaffey J., Cannady J., "Multiple Self-Organizing Maps for Intrusion Detection". Proceedings of the NISSC 2000 conference.
[9] Girardin L., "An Eye on Network Intruder-Administrator Shootouts". Proceedings of the Workshop on Intrusion Detection and Network Monitoring, Santa Clara, CA, USA, April 9-12, 1999.