Multiple Self-Organizing Maps for Intrusion Detection

Brandon Craig Rhodes, James A. Mahaffey, James D. Cannady
{brandon.rhodes,jim.mahaffey,james.cannady}@gtri.gatech.edu

Computer Science and Information Technology Division
Information Technology and Telecommunications Laboratory
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

While many techniques have been explored for detecting intrusive or abnormal behavior on computer systems, approaches that involve pattern matching, expert systems, and traditional neural networks require detectors to either be crafted by hand or trained upon examples of known intrusions. We argue that neural networks capable of unsupervised learning can provide a powerful supplement to these techniques. After learning the characteristics of normal traffic or user behavior, these networks can identify abnormalities without relying on expectations of what abuse will look like. This paper analyzes the potential of the Kohonen self-organizing map to narrow the envelope of intrusive behaviors that would not be caught by a detection system.[1]

Keywords: Intrusion detection, misuse detection, neural networks, Kohonen self-organizing map.

1 Introduction

Intrusion detection systems have been an active area of research and development since 1987 [Den87]. Increases in speed and complexity of computer networks, as well as public access to them, have sharpened the need for effective intrusion detection approaches. This paper describes an innovative approach to intrusion detection that uses self-organizing neural networks to recognize anomalies in a computer network data stream. Unlike most existing approaches, these networks could recognize potentially intrusive activity in network traffic without utilizing the pre-defined attack signatures that limit the effectiveness of typical approaches using expert systems. The following sections provide an overview of existing intrusion detection techniques, a discussion of neural networks and their application to intrusion detection, and a description of the ongoing research effort that is applying self-organizing neural networks in the detection of network attacks.

[1]This work is supported by Army Research Laboratory DAKF11-97-D-0001-0036, and is the subject of a provisional patent application.

2 Prior Neural Network Approaches

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 skill of many attackers combine to create a problem domain that is extremely difficult to address.

Intrusion detection has traditionally focused on one of two approaches. *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. 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. 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-base in an attempt to compensate for this weakness, with a side effect of reducing the granularity of the intrusion detection process.

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 system users and identify statistically significant variations from their established behavior.

Fox employed a Kohonen self-organizing map [FHR90] to learn the characteristics of normal system activity on a multiuser machine and identify variations from that norm that could indicate the presence of a virus. Like us he included in the map's output not only the location to which input has been assigned, but the measure of how well it fits there — and like us made the latter factor primary in his analysis.

The first use of a Kohonen self-organizing map in misuse detection is described by Cannady [CM98]. There a hybrid neural network — in which the output of a Kohonen map provided input to a conventional feed-forward neural network — was prototyped to address temporally dispersed, and possibly collaborative, attacks in a simulated data stream. Temporally dispersed attacks are those conducted by a single attacker over an extended period of time, while collaborative attacks are conducted by multiple attackers working in concert to achieve a single intrusion. Each of the attackers' actions taken individually may appear innocuous, the attack becoming apparent only if all of the events are considered together. Both of these types of attack may be very complex. To address this, an enhanced pattern recognition approach was designed and tested to identify series of events that constitute an attack. The tests conducted on the hybrid neural network indicated that the prototype was very effective in identifying the simulated attacks.

While earlier work considered Kohonen maps that processed the entire state of a system or network, our work breaks ground by using collections of more specialized

maps. By introducing deterministic preprocessing of network traffic, we narrow the load placed on each map and, we argue, facilitate the construction of a system that is much more sensitive to abnormalities in these separate streams of network traffic.

3 The Kohonen Self-Organizing Map

Typical neural networks, to which most readers will have been exposed, can only be trained after their teacher has determined what output he desires for each of many possible input values. In contrast, the *self-organizing map* developed by Kohonen [Koh95] automatically categorizes the varieties of input presented during training and can then express how well new inputs fit the patterns it has discerned.

3.1 Kohonen Map Structure

An input to a Kohonen map takes the form of a mathematical *vector* (an ordered list of coordinates). The map must have been provided with some means of measuring how different two vectors are; for this paper we will call this function that compares two vectors our *comparison function.* Later in the paper we will consider several possible comparison functions.

The map itself is two-dimensional — a surface across which a grid of prototype vectors is laid (the regularly spaced dots across the maps in Figure 1 indicate the configuration of each grid). We have been using square maps, each covered by a uniform rectangular grid of prototype vectors, but other options do exist (hexagonal maps are, for instance, sometimes encountered). We use the term *prototype* for the vectors at the intersections of the grid to make it clear that the vectors on the grid are not copied directly from the input, but represent ideal or average vectors of which many examples may exist in the map's input.

When we present an input vector to the map for evaluation, it is compared (using the map's comparison function) with every prototype vector in the grid. The map then outputs two pieces of information: the location (in grid coordinates) of the prototype that most closely matches the input vector, and the measure of how well the input vector fits the prototype (that is, the metric returned by the comparison function). If we desire more detail in the locations returned by the map, we can instead have the map interpolate a location between the two or three most similar prototypes; in fact for our experiments we use two-point interpolation.

Of course, we are not usually interested in vectors for their own sake, but use them to describe the properties of some other class of objects which we wish to study. If we were, for example, interested in the triangles in Figure 1a, we would create a vector for each triangle that described its shape. This was in fact done (the caption describes the design of the vector), and to understand the result we must study how the Kohonen map is trained.

3.2 Kohonen Map Training

Given a set of input vectors on which to train, the Kohonen map spontaneously organizes them so that similar vectors are clustered together on its surface, and that important variations among vectors are reflected in their arrangement. The distance between where two vectors are placed on its surface represents not their absolute dissimilarity — which we already knew how to compute — but their difference relative

to the space of possibilities present in the training set. Figure 1b illustrates such an arrangement: each triangle has been drawn where the map placed the vector describing its properties, and we see that the two types of triangles have been segregated — with all the fat equilaterals piled up in one cluster — and the isosceles triangles have been stretched along arcs that explore the fine distinctions among their orientations.

The training process through which these relationships are established are fairly straightforward. Choose a set of vectors upon which to train the map, and randomize the map's prototype vector elements within the range of values that the corresponding input vector elements are expected to offer. Then iterate over the training vectors; for each training vector, find the prototype on the map which is most similar to it, and incrementally adjust that prototype and its neighbors on the map to more closely resemble the training vector. Repeatedly subject the map to the training vectors this way, gradually reducing the size of both the 'neighborhood' that gets adjusted and the increment by which elements are adjusted.

To prevent the map from clumping the training vectors on only a few favored prototypes, a bias against winning prototypes can be introduced to force the training vectors to make more extensive use of the map's area.

4 Kohonen's Map and Anomaly Detection

Given the Kohonen self-organizing map's ability to categorize a collection of input vectors then rate whether subsequent vectors fit any of those categories, we need only a technique for turning network traffic into vectors to subject that traffic to the same type of analysis.

4.1 Constructing a Monitor Stack

Inspection of the network link serving an Internet-connected facility will typically find it packed with heterogeneous traffic. There are several sources of this variety:

- Packets are not only destined for different hosts, but belong to particular conversations ongoing between applications on those hosts. Other packets may not belong to conversations, instead carrying asynchronous unidirectional communiques that receive no acknowledgement.
- Each packet bears a series of headers that each direct the operation of one of the protocols which the communicating agent is employing to transmit his information.
- The pace and duration of connections is quite variable. Automated transactions such as nameserver and HTTP requests may often take less than a second, while login sessions operated directly by humans may persist for hours.
- The payloads carried by the observed packets will vary remarkably between applications. Some will hold exactingly formatted text, others encode the painstaking manual edits of users at their keyboards, and still others will communicate arbitrary binary data.

It would be unreasonable to expect a single Kohonen map to usefully characterize such disparate information.

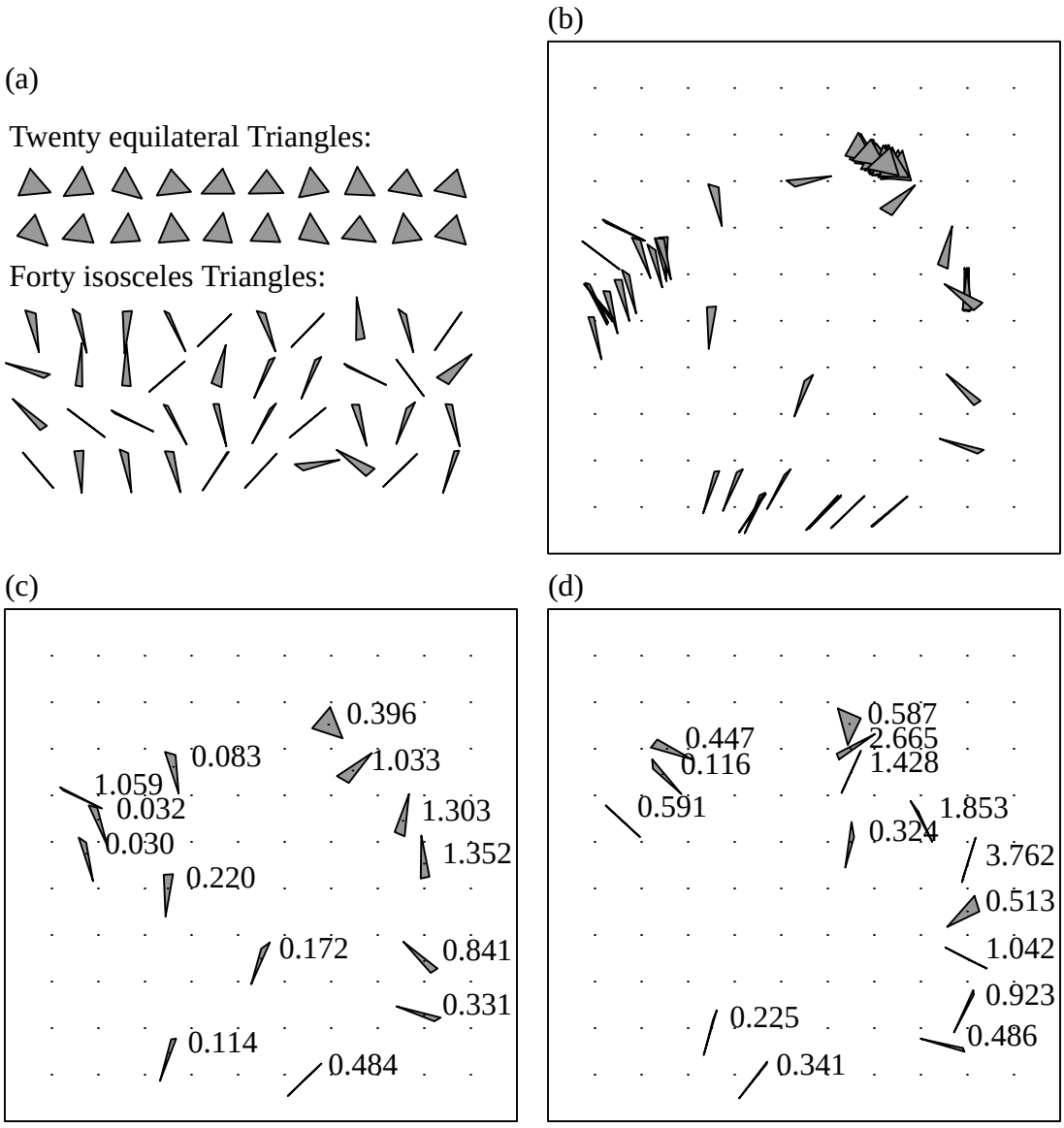

Figure 1: **(a)** Sixty random triangles, deliberately generated within two narrow categories, form the training set for an example Kohonen map. **(b)** This ten-by-ten Kohonen map was trained upon vectors which each describe the area, acuteness, and orientation of one triangle from the training set; here each triangle is shown at the position assigned to its vector by the self-organizing map. **(c)** A subset of the training triangles is shown (for clarity), each labelled with the measure of how different its vector is from the map element to which it is most similar. **(d)** Finally we use the map for its intended purpose — categorizing novel inputs; here a random assortment of triangles, some similar to the trainee vectors and some different, are shown in position and labelled with how great a difference they register compared with the underlying map elements.

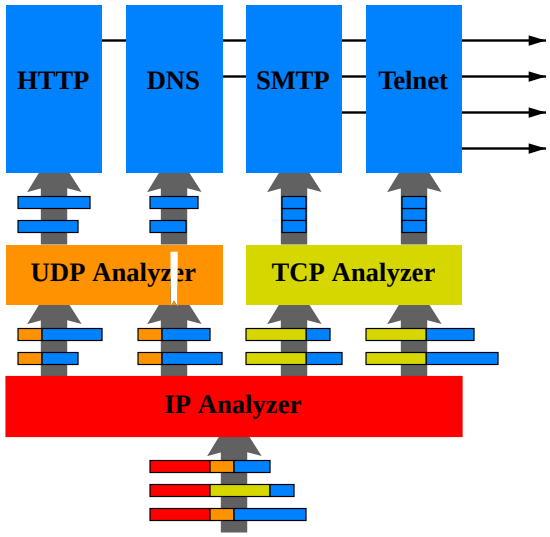

Figure 2: General schematic for a network monitor stack, including suggestions of the kinds of information available at different levels.

These considerations motivate the construction of a *monitor stack*, utilizing protocol analyzers to reduce and segregate the traffic before it is subjected to map analysis. Figure 2 presents a schematic monitor stack for interpreting Internet Protocol (IP) traffic. Since packets are produced by layered protocols in the originating host, the monitors are similarly layered — allowing the extraction of statistics about lower protocol layers while permitting the reconstruction of the data streams they support.

Note that activity at every layer of the monitoring stack must be monitored, since malicious activity can be targeted at *any* protocol layer. An otherwise innocent web browser request could be degenerately fragmented at the IP layer in an attempt to crash the receiver's operating system; floods of apparently normal connection requests could be targeted to monopolize the resources of a server's Transmission Control Protocol (TCP) layer to deny service to legitimate users; or, an abnormally long and garbled file name requested from a file server could induce it to run arbitrary code on behalf of an intruder.

Designing an analyzer stack for reconstructing network activity is not novel, but is an essential component of a neural network approach. By illustrating the kinds of information available from a complete decomposition of network traffic, it provides a context in which we may consider the intelligent deployment of neural networks.

4.2 Specialist Networks

The monitor stack architecture strongly suggests that each neural network become a kind of specialist, trained to recognize the normal activity of a single protocol and ready to raise an alarm when a significant deviation is detected. The specialization could be carried even further, with the activities of particular remote hosts or individual users being subjected to the analysis of neural networks honed to recognize their typical behavior.

Of course many desirable monitors can, and should, be constructed without neural

networks — many such schemes are already operating in extant intrusion detection systems. It would be ridiculous to attempt a neural approach to recognizing, say, an attack based on the reassembly of malformed IP fragments, when simple conditional checks on fragment offsets and lengths will accomplish this flawlessly and very efficiently. Rather, we are seeking out those roles in the network protocol hierarchy where flexible pattern recognition is a significant asset over more traditional approaches.

Note that protocol decomposition can continue for as long as we can invoke further deterministic rules to guide it. Not only could the data stream reconstructed from TCP packets be further interpreted as a Simple Mail Transport Protocol transaction, but the mail message itself could be processed for MIME encoded attachments that might include documents with macros. The computing resources available for monitoring a given network link will of course place limits on how much processing can be performed, but we expect that many facilities would find it worthwhile to configure certain monitors to drill down quite far into the structure of transactions that have posed earlier security problems.

4.3 Vectorization Options

Our discussion to this point has glossed over the issue of how incoming data streams are packaged for presentation to a Kohonen map, which (as was noted in the earlier discussion of them) accepts fixed-length vectors as input.

The invention of schemes by which packets, transactions, and data streams might be represented numerically represents both the greatest challenge and the greatest opportunity of this research. The choice of which traffic features to represent, and how to translate them into numbers, will unavoidably involve highlighting certain aspects of network activity while making others obscure or even invisible to the detector. The designer of each detector must therefore strive to develop vectorizations which increase, as much as is possible, the contrast between innocent and malicious activity.

The possibility that several successful vectorization techniques could be developed for a single protocol suggests the deployment of multiple Kohonen maps, each viewing that network activity through the lens of a different vectorization. While it may often be possible to construct an intrusion that appears normal to one particular map, the presence of several maps — each watching a different aspect of the protocol's behavior — will make it extremely unlikely that an intrusion could pass unnoticed.

An open issue which our research shall investigate is which features should be combined to make a single vector, and which should be presented to different maps in separate vectors. If too many features are combined into a single vector, we run the risk that an intrusion will not make enough of them suspicious make the vector as a whole appear anomalous; but if every feature is presented to its own map, then we lose the ability to detect unusual correlations. We hope in the course of our research to develop guidelines for detector construction that address these questions.

Another choice, just as important as the design of the vector, is the formula for measuring the similarity between two vectors. The comparison function used in Figure 1, for example, gave the greatest weight to the difference between the triangles' areas; thus it absolutely segregated the fat equilaterals and thin isosceles, despite the fact that there are other respects in which they are similar.

The most obvious measure of vector similarity is somehow aggregating the differences among their several coordinates. But if vector coordinates represent actual content, or information for which absolute order is not important, other techniques might be used for comparison. We might measure the differences between two texts using the (Levenshtein) editing distance; the similarity between parts of documents with Tanimoto similarity; or statistical pattern recognition. While these techniques tend to be quite expensive and would therefore not usually be appropriate for packet analysis, they might prove useful when comparing transmitted email messages or other large documents.

It must finally be admitted that many vectorization techniques will be suited for detecting some intrusions at the expense of others (though further research may teach us how to construct vectorizations that are quite general). At this point our approach may prove susceptible to one weaknesses of the rule-based approaches with which he have contrasted it: that it only allows the detection of attacks that we already know about. So while a Kohonen map designed with buffer overflow attacks in mind will almost certainly be more flexible than a set of rules watching for the same type of activity, both may completely overlook some other kind of attack of which their designers were ignorant. The Kohonen map may thus have a limit to how far outside our expectations it can detect abnormal activity.

4.4 Over-specificity and Retraining

The activities of networks and their users change slowly over time; new areas of application can produce new traffic on old protocols and services, while changes in server and client behavior gradually shift the content of sessions and transactions. This presents both a challenge and opportunity for neural networks.

The challenge can be termed the problem of *over-specificity*. We will presumably train the Kohonen map on data collected from the site to be monitored. This means that instead of creating a map that recognizes, say, web traffic in general, we will be creating a monitor that specifically accepts the web requests common at a particular installation. As the web content at that site evolves, portions of the traffic it produces may begin to appear abnormal to a map trained months earlier.

But if indeed map training can be an unsupervised process, as described elsewhere in this paper, then this problem invites periodic, automatic *retraining* as its solution. At various intervals the intrusion detection system can capture intervals of traffic and use them to train new networks. The old network could either be immediately discarded, or several generations could be kept online for comparative evaluation and the oldest map rotated off once its false-hit rate became unbearable.

4.5 Additional Research Foci

In the course of our initial research, two challenges have arisen whose resolution will be key to successful implementation of our technique. Here we provide an outline overview of each challenge.

First, there appear to be specific situations in which the measure of how well a vector fits on the Kohonen map is by itself an insufficient indicator of whether that vector is in fact representative of the input. An example occurs when the input falls naturally into two categories with widely different attributes: the map will carve out

an area to fit each category, leaving between them an empty region in which the prototypes represent intermediate cases between the two observed extremes. If the map then receives such an intermediate vector as input, it will place it quite comfortably in that middle region despite the fact that it resembles none of the input vectors. There are at least two approaches to disambiguating these situations: first, we could contrive some method of recording which areas of the map were occupied by training vectors and considering vectors suspicious that were outside of these regions; or, we could seek some deterministic means of distinguishing the two types of traffic, and train two separate Kohonen maps upon the properties of each.

Another issue involves the collection of training examples — we have blithely been assuming an ability to acquire arbitrary amounts of intrusion-free traffic. At a real computing installation intrusion attempts from the Internet might happen at any time, and popular sites might find them so frequent that any significant collection of captured traffic is likely to include one or two. Of course we can use a traditional expert system to remove known intrusions from our data, but the whole point of our exploring a neural network approach is that such systems are insufficient because novel intrusions are always being developed. If such a novel intrusion is present in our training data, will we not inadvertently create a Kohonen map specifically designed to *accept* that intrusion as normal network behavior?

We have identified two approaches toward mitigating the problem of intrusions in the data set. One involves the fact that, given its size and the strength of the neighborhood function used in training, a given self-organizing map possesses a limit on how much diversity can exist among the vectors it accepts. As long as the intrusion occupies only a fraction of the training set, and generates sufficiently distinct vectors, the map will be dominated by normal activity and the intrusion will not be able to carve out a region that accepts it. Maps that are trained using possibly tainted input should be checked for data that was not able to be accommodated.

A more robust approach to removing anomalies from training data is to collect several training sets, train a map on each set, then filter each set through the other maps. Only if similar intrusions manage to appear in *every* data set will they not be recognized by one of the maps. Once cross-filtering has been used to clean the training sets, one or any of them can be used for training a final map.

5 Experimental Evidence

We have constructed an anomaly detector based on a Kohonen map, and successfully detected two different exploit attempts we perpetrated against our own server. The following is an outline of our experiment.

5.1 Design

We have implemented a Kohonen map using the C language on a Linux workstation, and a test harness written in Python that includes basic visualization facilities allowing us to monitor the training and use of the map. All of our network data was collected locally using the *tcpdump* packet capture program that is standard equipment with many versions of Unix.

We chose to monitor requests to our Domain Name Service (DNS) port because of the relative simplicity of its protocol: its requests and replies are defined to be symmetric, so a single routine can be used to parse both. DNS implements a worldwide distributed database for performing host name resolution, allowing users to use names like "mit.edu" rather than raw IP addresses like 18.72.0.100. We restricted our attention to requests using the TCP protocol, since this is the protocol used by both of the exploits which we possess.[2]

From a sample of roughly forty packets we used the first thirty to train the map, and the other dozen to determine how well the map generalized from these thirty examples. A listing of all forty vectors, sorted by how well they fit on the map, showed that the map did not noticeably favor the thirty vectors from the training set. We then submitted the packets from both exploits to the trained maps and measured how well they fit.

Since both of the test intrusions were buffer overflow attempts, we designed vectors consisting of a simple six-category histogram indicating how many octets each packet fit a particular character class (such as alphabetic, numeric, control, and non-ASCII). Since buffer overflow arguments tend to be unusually large and place binary data where text is expected, we hypothesized that this vectorization scheme would make buffer overflow attempts particularly striking.

Note that we have not at this point attempted to reconstruct the TCP packets into a data stream, as the system envisioned above would do. Since we instead simply consider each packet individually, our results may be more modest than would be achieved by a more complete monitor stack.

5.2 Results

We were extremely encouraged by the stability of our self-organizing map. Since each map is initialized randomly, and reaches a unique spatial arrangement for the training vectors, we did not know whether our measure of how well each vector fit would also vary substantially. Our training runs have to date produced quite consistent measures, rarely approaching a ten percent difference between the measures of fit for the same vector on different runs. So while the location of the clusters is different each run, the properties of the topological habitat they carve out are eminently reproducible.

In the case of our data and our particular distance measure, all of the "normal" traffic scored somewhere between zero and three, save for one outlier that consistently registered around thirteen — not a very good fit.

The bind4-9-5 exploit produced encouraging results. Of the roughly seven packets transmitted to accomplish the exploit, two registered just above eighty, indicating they did not fit well on the map at all, and two other registered around six hundred thirty — indicating an extreme anomaly. This ratio of more than fifty between the worst-fitting training vector and the vectors produced from intrusive packets would provide ample leeway for establishing an alarm threshold in production system.

The results from the rotshb exploit were even more extravagant; not only did it also display two packets whose fitness measure landed at just above eighty, but the

[2] We employed both the rotshb (Riders of the Short Bus) and bind4-9-5 exploits, which can be downloaded from the Packet Storm security archive (visit http://packetstorm.securify.com/exploits/apps/bind/).

two packets actually carrying the exploit payload scored a whopping thirteen hundred and sixty each!

These experiments provide strong encouragement for the implementation of a working intrusion detection system utilizing self-organizing maps.

6 Conclusion

The Kohonen self-organizing map is an extremely powerful mechanism for automatic mathematical characterization of acceptable system activity. We have argued that it should be applied to the analysis of data collected from network monitoring, and have designed a monitoring system that would preprocess network packets to highlight their properties for inspection by a self-organizing map.

Our actual experiments show that even a simple map, when trained on normal data, will detect the anomalous features of both buffer overflow intrusions to which we exposed it. The ratio by which normal and intrusive packets differed in the self-organizing map was computed, and found to be greater than an order of magnitude in both instances. Note that buffer overflows are considered the most significant practical security threat of the last decade [CWP+99].

This approach is particularly powerful because the self-organizing map *never needs to be told* what intrusive behavior looks like. By learning to characterize normal behavior, it implicitly prepares itself to detect any aberrant network activity.

References

[CM98] James Cannady and Jim Mahaffey. The application of artificial intelligence to misuse detection. In *Proceedings of the First Recent Advances in Intrusion Detection (RAID) Conference*, 1998.

[CWP+99] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *Proceedings of DARPA Information Survivability Conference and Expo*, 1999.

[DBS92] H. Debar, M. Becke, and D. Siboni. A neural network component for an intrusion detection system. In *Proeedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1992.

[Den87] Dorothy Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2), 1987.

[FHR90] Kevin L. Fox, Rhonda R. Henning, and Jonathan H. Reed. A neural network approach towards intrusion detection. In *Proceedings of the 13th National Computer Security Conference*, 1990.

[Fra94] Jeremy Frank. Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th National Computer Security Conference*, 1994.

[Koh95] T. Kohonen. *Self-Organizing Maps*. Berlin: Springer, 1995.