Collaborative Intrusion Detection System

Patrick Miller* and Atsushi Inoue**
Department of Computer Science
Eastern Washington University
Cheney, WA 99004-2412 U.S.A

Abstract

This paper presents an intrusion detection system consisting of multiple intelligent agents. Each agent uses a self-organizing map (SOM) in order to detect intrusive activities on a computer network. A blackboard mechanism is used for the aggregation of results generated from such agents (i.e. a group decision). In addition, this system is capable of reinforcement learning with the reinforcement signal generated within the blackboard and then distributed over all agents which are involved in the group decision making.

Systems with various configurations of agents are evaluated for criteria such as speed, accuracy, and consistency. The results indicate an increase in classification accuracy as well as in its constancy as more sensors are incorporated. Currently this system is primarily tested on the data set for KDD Cup '99.

1. Introduction

Intrusion detection systems have become important components of major network security systems. The goal of such systems is to identify potential violations of the network security policies. To this end, a variety of data may be used as the input, whether it is TCP/IP traffic, kernel calls, or system logs. Regardless of the nature of the data two principal approaches may be included in these systems: *anomaly detection* and *signature-based detection*[1].

Both systems have their advantages, and indeed many systems, including the one presented here, use hybrid methods.

A computer network can come under attack from a variety of sources, and a variety of methods. As would be expected, such attacks have massively varied signatures as well. Consequently, a multi-agent approach is studied; such that each agent detects a specific attack type and decisions made by the agents are aggregated as a decision of the entire system. This approach is studied because it may be more feasible than attempting to design a single complex system that detects all forms of attacks.

In order to obtain useful patterns of attacks, it is necessary to collect a significant number of samples. Unfortunately, such a complex problem as intrusion detection does not always allow us to collect the significant number of samples (a.k.a. base-rate fallacy problems) [2]. On the other hand, a cognitive study shows that patterns generated with subjective estimates based on frequencies closely overcome these generated with normative predictions in classification tasks [7]. Soft computing (SC) approaches are the most suitable for handling such subjective estimates [8].

In this paper, a multi-agent intrusion detection system consisting of heterogeneous agents using self-organizing maps (SOM) [4] and ordinary rule-based classifiers is sought. These agents perform pattern classification by comparing each input against a stored map, finding the closest approximation to that input and updating the map to reflect the new data. This has the effect of clustering inputs into groups, the assumption being that similar inputs will generate similar results on the target system. To improve the classification rate for each type of attack, each agent has its own learning parameters and analyzes the same input simultaneously. The combined result of these agents provides a more reliable output than that generated by a single agent.

All agents are configured within a three-stage architecture such that the first stage performs signature-based detections for well studied intrusions using ordinary rule-based classifiers. The second stage uses an array of SOM-based agents in order to perform anomaly detection, and the third stage

* E-mail: patrick@doriathproject.com
** Director of the Inland Northwest Security Systems Initiative (INSSI) within the Department of Computer Science at Eastern Washington University. Research partially supported by the Congressional Appropriation of Technology Initiative for the New Economy (TINE). E-mail: atsushi.inoue@ewu.edu

detects specific patterns from abnormal activities for further analysis on new signatures of attacks using other SOM-based agents. Preliminary experiments have indicated that this multi-agent system with the three-stage architecture detects even rare attacks amidst a large data set and thus mitigates the effects of the base-rate fallacy problem.

Such an autonomous intrusion detection system should be measured on a number of criteria – security, inter-operability, transparency, accuracy, speed, scalability, and ease of use. An application framework introduced in this paper is measured with a focus on these factors, particularly on speed, accuracy, and ease of use.

2. General Framework

The Synergistic and Perceptual Intrusion Detection with Reinforcement (*SPIDeR*), a general intrusion detection framework that consists of multiple autonomous agents with heterogeneous computational models, is currently begin studied [6]. Those agents are distributed over a computer network, perform intrusion detection tasks autonomously and asynchronously. Their results are stored in a single blackboard agent and are aggregated by a decision-making agent.

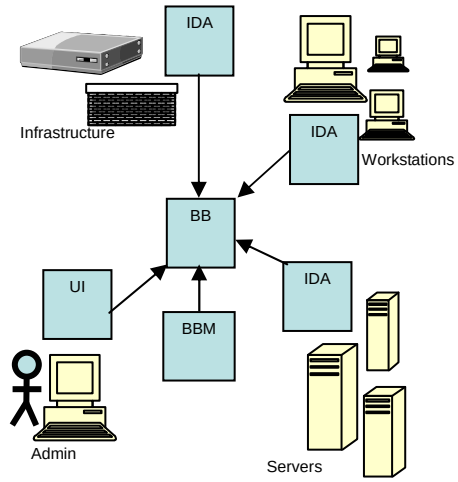

Figure 1. SPIDeR System Architecture

2.1. Architecture

SPIDeR consists of the following agents shown in Figure 1:

1. Blackboard (*BB*) maintains information regarding intrusion detections in the computer network.
2. Blackboard management subsystem (*BBM*) has two tasks: the aggregation of decisions made among agents and housekeeping.
3. User interface (*UI*) provides visualization of information stored on BB and an interface for the administrator to configure intrusion detection operations.
4. Intrusion detection agent (*IDA*) autonomously and independently performs its intrusion detection task.

Computational models of IDAs vary, e.g., fuzzy logic, neural networks, probabilistic reasoning, support logic and string matching based on regular expressions.

2.2. Decision Making Model

Within SPIDeR, the system administrator and agents collaboratively make decisions as to whether or not intrusions are detected. All decisions made by those agents and the system administrators are recorded on BB. BBM then aggregates them in order to determine the most appropriate actions. Consider a set of decisions D corresponding to all intrusions and normal cases (i.e. no intrusions). Then the following decisions are made by IDAs:

1. *Crisp decision:* a particular decision x_c in D is made (i.e. probability $P(x_c) = 1$).
2. *Probabilistic decision:* a decision x_p in D associated with a point probability $P(x_p)$ such that $P(x_p) + P(\text{not } x_p) = 1$ is made.
3. *Support decision:* a decision x_s in D associated with a support pair, i.e. an interval of probability, $(P_l(x_s),P_u(x_s))$ where $P_l(x_s) \le P_u(x_s)$ is made. Consequently, the support pair for the complement is determined such that $(P_l(\text{not } x_s), P_u(\text{not } x_s)) = (1-P_u(x_s), 1-P_l(x_s))$.
4. *Fuzzy decision:* a decision x_f in D associated with a fuzzy probability $P_f(x_f)$, a fuzzy set defined over [0,1], is made (e.g., $P_f(x_f)$ = 'high' where 'high' is a fuzzy set defined over [0,1].)

Then the team decision x associated with a probability $P(x)$ is obtained by aggregating $P_i(x)$ where $i = 1 \ldots n$ (i.e. the probability that the i-th agent makes decision x) such that

$$P(x) = \mathcal{H}_w(P_1(x), \ldots, P_n(x)) \qquad (1)$$

where w is the vote weight (usually normalized, i.e. $\Sigma_i\ w_i=1$) representing the influence on the decision making. $\mathcal{H}$ is a notation indicating a generic

aggregation combination operation such that $\mathcal{H}$: $[0,1]^n \rightarrow [0,1]$.

2.2.1. Point Combination

The aggregation of decisions can be performed by a simple weight averaging combination such that

$$P(x) = \Sigma_i (w_i \bullet P_i(x)) \qquad (2)$$

where $\Sigma_i\ w_i = 1$ (i.e. the summation of normalized votes). Support decisions and fuzzy decisions can also be combined if they are represented as point probabilities. For instance, a support decision is represented such that $P_i(x)=(P_l(x)+P_u(x))/2$ and a fuzzy decision is represented by a defuzzification such that $P_i(x) = (\Sigma_{y\in[0,1]} \mu P_i(y) \bullet y)/(\Sigma_y \mu P_i(y))$.

2.2.2. Support Combination

Alternatively, we are currently studying a more generic combination operation utilizing Mass Assignment Theory (MAT) [3]. MAT provides a framework for aggregating multiple decision types that are made by a set of agents. More formally, let D_j be a set of decisions that the j-th agent makes. Note that $D_j \subseteq D$ and $\cup_j D_j = D$ (D is a set of decisions made by an instance of SPIDeR). Suppose that the j-th agent has a normalized vote w_j (i.e. $\Sigma_{j\varepsilon\{1\ldots n\}} w_j = 1$). Then the probability that a decision $x \in D$ is made by a team within SPIDeR is given by

$$P_D(x) = \sum_{D_j \subseteq D, x \in D_j} PD_j(x) \bullet w_j \qquad (3)$$

$P_{Dj}(x)$ is a probability associated with a decision made by the j-th agent. This probability can be obtained as follows:

1. *Crisp Decision:* $P_{Dj}(x)=1$ and $P_{Dj}(\text{not } x)=0$
2. *Probabilistic Decision:* $P_{Dj}(x)=p$ and $P_{Dj}(\text{not } x)=(1-p)/(|D_j|-1)$ assuming no bias.
3. *Support Decision:* Consider the following procedure:
   (a) Generate MA m_j corresponding to a support pair (p_l, p_u) for a decision x such that

$$\begin{aligned} &m_j(\{x\}) = p_l \\ &m_j(\{\text{not x}\} = D_j - \{x\}) = 1 - p_u \\ &m_j(D_j) = p_u - p_l \end{aligned} \qquad (4)$$

   (b) Compute $P_{Dj}(x)$ from m_j using MAT. Assume the least prejudged distribution $1/|A|$ where A is a focal element of m_j, for the selection rules unless it is given.
4. *Fuzzy Decision:* Let F_p be a fuzzy probability such that $P_{Dj}(x) = F_p$. Using MAT and the representation (decomposition) theorem, we first obtain a collection of F_α, a crisp set (so-called α-cut) consisting of elements x such that $\mu_{Fp}(x) \geq \alpha$ (i.e. an interval within [0,1]) and then MA m_{Fp} corresponding to F_p such that

$$m_{Fp}(F\alpha_i) = \alpha_i - \alpha_{i+1} \qquad (5)$$

   where, without loss of generality, α is sorted in non-increasing order such that $1=\alpha_1 \geq \ldots \geq \alpha_n \geq \alpha_{n+1}=0$ and F_α are the only focal elements for m_{Fp}. This leads to a collection of possible support decisions such that

$$(p_l = \text{MIN}[F\alpha_i],\ p_u = \text{MAX}[F\alpha_i]) \qquad (6)$$

   is true. Lastly, MA m_j for the collection of possible support decisions from F_p is obtained such that

$$\begin{aligned} &m_j(\{x\}) = \Sigma_i\ p_{li} \bullet m_{Fp}(F\alpha_i) \\ &m_j(\{\text{not x}\} = D_j - \{x\}) \\ &\qquad = \Sigma_i (1 - p_{ui}) \bullet m_{Fp}(F\alpha_i) \\ &m_j(D_j) = \Sigma^n\ (p^i - p^i) \bullet m_{Fp}(F\alpha_i) \end{aligned} \qquad (7)$$

   then $P_{Dj}(x)$ is computed from m_j by MAT.

2.3. Reinforcement Learning

Reinforcement learning[21] in SPIDeR takes place within the entire system in such a way that the reinforcement signal (either a reward or a penalty) obtained as a consequence of an action is distributed among IDA's with respect to their voting weights (see Figure 2). A computational outline follows:

1. A system administrator performs a certain action for decision $x\ \varepsilon\ D$.
2. For decision x, determine its reward $r(x) = \gamma \bullet (1 - P_D(x))$ where $\gamma \in [0,1]$ is a constant (i.e. a learning rate).
3. For other decisions x_i, where $x_i \neq x$, determine its penalty such that

$$r(x_i) = -r(x) \bullet \frac{P_D(x_i)}{\Sigma_{i,x\_i \neq x} P_D(x_i)}$$

4. Update the probability $P_D(x_i) \rightarrow P'_D(x_i)$ for all $i \in \{1 \ldots n\}$ such that $P'_D(x) = r(x_i) + P_D(x_i)$.
5. Obtain the corresponding updated (normalized) vote $w_i \rightarrow w'_i$ such that

$$P_D(x) = \Sigma_{D_j \subseteq D, x \in D_j} P_{D_j}(x) \bullet w_j$$

6. Determine rewards or penalties $r_i^w = w_i' - w_i$ for all decision makers.
7. Propagate r_i^w to all adaptive IDAs.
8. Adjust the vote w_i if the *i*-th IDA has adapted itself to improve performance.

If the agent is capable of handling reinforcement learning in addition to the above system bias it may update itself internally using whatever methodology it has implemented. The algorithm for reflecting those awards may vary depending on the computational models implemented within those agents. It can then notify the blackboard or its update, and the blackboard may choose to readjust that agent's trust level.

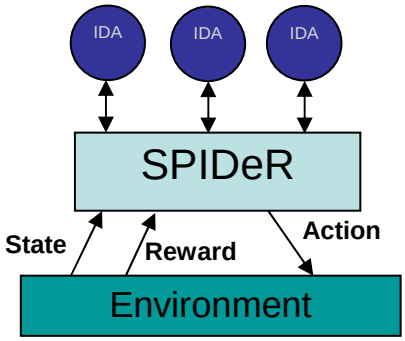

Figure 2. SPIDeR Reinforcement Learning

3. Specific Implementation Using SOM

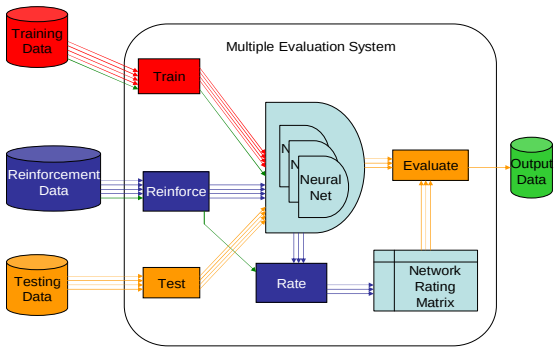

Figure 3. Overview of SPIDeR-MAN

The Synergistic and Perceptual Intrusion Detection with Reinforcement in a Multi-Agent Neural Network (*SPIDeR-MAN*) shown in Figure 3 is an instance of SPIDeR that utilizes agents based on self-organizing maps (SOM) and a three-stage architecture. It reads inputs containing a mixture of network traffic, system logs, and historical information. These inputs are processed by SOM-based agents. Each agent acts independently and is initialized with random values. This causes each agent to learn the patterns slightly differently. The responses generated by these networks are probabilistic decisions. These decisions are then combined by a point combination that uses a weighted voting scheme. These votes are determined based on performance during training. These votes are also updated at run time based on the confidence of the majority vote.

3.1. Self Organizing Maps

A self-organizing map (SOM) used in SPIDeR-MAN consists of an n-dimensional array (map). Each node of the map contains either an input value, or a compressed form of the input. These values may be randomly initialized or preset with domain specific knowledge. Additional configuration values may be set at initialization time to control the rate that the SOM adapts to and learns new inputs. During operation, the SOM reads in an input and uses a distance function to determine which map node the input most closely matches. The SOM then modifies that node, and, to a lesser degree, the nodes within a specified radius, to more closely match the current input. This incremental update in conjunction with the radius allows the SOM to generate clusters of related inputs. The output for the SOM is then either the cluster, or clusters, that most closely represent the current input. More specifically, the SOM algorithm is given as follows:

Step 1: (Do in parallel)
1.1 Initialize weights w_{ij}
1.2 Set topological neighborhood parameters
1.3 Set learning rate parameters
End in Parallel
Step 2: While stopping condition is false, do Steps 3-8
Step 3: For each input vector x, do Steps 4-6
Step 4: For each j, compute: $D(j)=\Sigma_i(w_{ij} - x_i)^2$
Step 5: Find index J such that D(J) is a minimum
Step 6: For all units j within a specified neighborhood of J, and for all i

$$w_{ij}(new)=w_{ij}(old)+\alpha[x_i - w_{ij}(old)]$$

Step 7: Update learning rate
Step 8: Test stopping condition

Although a SOM is inherently an unsupervised learning technique, there are conditions where a supervised technique would be more appropriate. Fortunately, SOMs can also behave like supervised learning systems through a variety of methods. One method is to use domain specific information to configure the initial values of the SOM so that they closely match the final values, and to only use the SOM for fine-tuning and to compensate for the gradual drift of data over time. Another quasi-supervised method is to allow the SOM to train itself to perform the clustering in an unsupervised fashion, and then to be provided with some supervised data to label the clusters. This is useful for making the SOM's output easier to interact with and more human readable.

The method most explored in our research allows the SOM to be fully supervised during a training period, but it limits the classification to binary decisions. With this method, multiple SOMs are used. During the training period, a given SOM is only fed the training data that matches with its particular pattern type. Such SOMs can be trained using smaller datasets, and are more likely to classify rarely seen patterns in a large dataset accurately. In this implementation, each SOM is essentially a single cluster. This gives the overriding system and the user greater control over the classification of a given type.

3.2. Decision Making Model

Such SOMs provide a flexible, easy to use, method for classifying input vectors. Although the system is flexible, any particular set of initial learning conditions may not be equally suited to learn a given pattern. Indeed many patterns are not linearly separable and may not be accurately classified by any single SOM.

To attempt to overcome this potential deficiency, multiple, independent SOMs, each with unique learning parameters, are used collaboratively in order to generate a more reliable classification framework. The results from each sensor agent are stored on the blackboard system. The blackboard also contains data indicating each agent's past performance for each type of attack. This data is used as a bias against each agent's vote. The biased votes are summed and the classification with the most votes is chosen as the final decision of the system. An overview of decision making in SPIDeR-MAN is shown in Figure 3.

3.3. Reinforcement Learning

For each input in the training process, each SOM is asked to evaluate the most recent training input. The results of this analysis are used to generate a bias. More specifically, the reinforcement learning of SPIDeR-MAN is given as follows:

```
Given Board B with n agents, and
that each input i in the training
file has a class C.
Then for each i:
  For each n:
    R=Bn.analyze(i)
    if (R==C)
       Totalnc++
    else
       Totalnc++
       Wrongnc++
       TotalnR++
       WrongnR++;
```

Sensors are assigned a trust level for each classification type. For example, if two sensor agents, A and B, analyze the same input and return different classification types, the result from the sensor with the highest trust rating for the type that it returned will be used. Once this training cycle has completed the SOMs are ready for evaluation. During this process, each SOM based sensor is passed an input vector, but is not allowed to see the classification field. The sensor then returns the classification that is the closest match to the input data. Additionally, since SOMs are an incremental learning system the SOM assumes that its guess was correct, and updates its map to reflect the new input. This ability allows the sensors to adapt to new variations of attacks.

The blackboard system compiles the results, as in Figure 3, returned from each sensor and applies the reinforcement signal that was assigned during training, to each sensor's response. These biased values are then combined to generate the most likely response. If the closest match is not within a given tolerance a new sensor node may be dynamically allocated.

4. Experiment

The following experiment has been conducted with the data set for KDD Cup '99[5]:

```
1. Execute program
   a) Initialize Board B
   b) Create n SOMs and inform B
   c) Train B
      i) For each input i in the
         training file:
```

- Train each *n* on *i*
- Test each *n* on *i*
- Update accuracy rating for each *n*

d) Test *B*
   i) Start timer
   ii) For each input *j* in the testing file:
   - Test each *n* on *j*
   - Compare each *n*'s response *r* to the correct class *c*.
   - If *r*=*c*
     - o Increment correct
     - o If r="normal" and c ⊆ attack
       a) Increment falseNormal
     - o If r ⊆ attack and c="normal"
       a) Increment falseAttack
   iii) Stop timer
e) Print results

4. Calculate averages

Table 1. Classification of the KDD Cup '99

	Spd	Acc	SDev	FA	FN[x]
1	60.26	78.52	12.53	13.15	4.797
2	37.98	85.66	2.302	11.56	1.611
3	18.04	85.33	3.839	13.30	5.155
5	8.786	87.66	1.493	10.08	1.872
10	5.628	86.51	2.130	11.09	1.974
20	2.700	87.70	0.992	9.763	1.916

Spd=Entries/Second
Acc=Correct/Total
SDev=Standard deviation of Acc
FN=(Normal reported when Attack)/Total
FA=(Attack reported when Normal)/Total
[x]--Value*10^{-2}

The results of these tests using one, two, three, five, ten, and twenty sensor agents on the same dataset are shown in Table 1. For consistency these tests were run on twenty identical PC's with 600MHz Pentium III processors and 512MB of RAM, each running Windows 2000.

By using randomly initialized SOMs we were assured a relatively low classification rate per agent. This was useful in showing the increase in accuracy when using multiple agents. By using multiple sensor agents in parallel to analyze the same data the accuracy can be increased. With the addition of one extra sensor, the testing results indicated an increase of over 7% in accuracy. The five-agent system was shown to classify correctly nearly 10% more events then the single sensor system. This indicates that although accuracy increases with the number of sensor nodes, this increase is not linear and tapers off dramatically after only a few sensors.

5. Conclusion

A framework of intrusion detection using heterogeneous multiple agents and SOM, SPIDeR-MAN, is discussed. A computational experiment is conducted by using the KDD Cup '99 data set. The results of this testing indicate an increase in system accuracy and consistency with the addition of multiple autonomous detection agents. We are planning to incorporate software sensors that capture data in real-time basis as well as other heterogeneous intrusion detection agents (e.g. SVM, ID3, Fuzzy Logic).

Acknowledgment

Gratitude is extended for the instrumental help of fellow researcher, Kristopher Smith. The authors also would like to recognize students working on the intrusion detection project in the machine learning class at EWU in Spring 2003.

References

[1] S. Axelsson, "*Intrusion Detection Systems: A Taxonomy and Survey*," Technical Report No 99-15, Dept. of Comp. Engr., Chalmers University of Technology, Sweden, 1999.

[2] S. Axelsson, "*On a Difficulty of Intrusion Detection*," RAID, West Lafiyette, IN, 1999.

[3] J. F. Baldwin, T. P. Martin, B. W. Pilsworth, *FRIL: Fuzzy and Evidential Reasoning in AI*, Research Studied Press, 1995.

[4] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice-Hall, 1994.

[5] KDD Cup 1999 data set, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[6] P. Miller, J. Mill, A. Inoue, "*Synergistic and Perceptual Intrusion Detection with Reinforcement (SPIDeR),*" MAICS, Cincinnati, OH, 2003.

[7] G. Gigerenzer and U. Hoffrage, "*How to improve Bayesian reasoning without instruction: Frequency formats.*" Psychological Review 102(4):684-704, 1995.

[8] L. A. Zadeh, "*Fuzzy Logic, Neural Networks, and Soft Computing*", Comm. ACM, vol. 37, no. 3, pp. 77-84, 1994.