Unsupervised Genetic Algorithm Deployed for Intrusion Detection

Zorana Banković[1], Slobodan Bojanić[1], Octavio Nieto[1] and Atta Badii[2]

[1]ETSI Telecomunicación, Universidad Politécnica de Madrid, Ciudad Universitaria s/n, 28040 Madrid, Spain
{zorana, slobodan, nieto }@die.upm.es

[2]Intelligent Media Systems & Services Research Centre School of Systems Engineering, White Knights campus, University of Reading, Reading, RG6 6AY, United Kingdom
atta.badii@reading.ac.uk

Abstract. This paper represents the first step in an on-going work for designing an unsupervised method based on genetic algorithm for intrusion detection. Its main role in a broader system is to notify of an unusual traffic and in that way provide the possibility of detecting unknown attacks. Most of the machine-learning techniques deployed for intrusion detection are supervised as these techniques are generally more accurate, but this implies the need of labeling the data for training and testing which is time-consuming and error-prone. Hence, our goal is to devise an anomaly detector which would be unsupervised, but at the same time robust and accurate. Genetic algorithms are robust and able to avoid getting stuck in local optima, unlike the rest of clustering techniques. The model is verified on KDD99 benchmark dataset, generating a solution competitive with the solutions of the state-of-the-art which demonstrates high possibilities of the proposed method.

Keywords: intrusion detection, genetic algorithm, unsupervised

1 Introduction

Software applications (both commercial and research ones) that deploy a machine learning technique are considered to be among the emerging technologies that have demonstrated compelling value in enhancing security and other related data analysis. Machine learning-based applications use complex mathematical algorithms that scour vast amounts of data and categorize them in much the same fashion as a human would, or at least according to the categorization rules set by a human. Yet, they are able to examine far more data in less time and more comprehensively than a human can, highlighting those events that appear suspicious enough to warrant human or automated attention.

Genetic Algorithms (GA) [1], in particular, offer certain advantages over other machine learning techniques, namely:

- GAs are intrinsically parallel, since they have multiple offspring, they can explore the solution space in multiple directions at once. If one path turns out to be a dead end, they can easily eliminate it and continue work on more promising avenues.
- Due to the parallelism that allows them to implicitly evaluate many schemas at once, genetic algorithms are particularly well-suited to solving problems where the space of all potential solutions is truly huge - too vast to search exhaustively in any reasonable amount of time, as network data is.
- Working with populations of candidate solutions rather than a single solution, and employing stochastic operators, to guide the search process permit GAs to cope well with attribute interactions and to avoid getting stuck in local maxima, which together make them very suitable for dealing with classifying rare classes, as intrusions are.
- A system based on GA can easily be re-trained. This property provides the adaptability of a GA-based system, which is an imperative quality of an intrusion detection system bearing in mind the high rate of new attacks emerging.

Our GA forms clusters of similar data and the principal idea is to allow the security expert to be the final arbiter of what is and is not an actual threat. In addition, after forming the initial clusters and assigning them the corresponding labels (also performed by the security expert), when classifying new events that do not correspond to any of the existing clusters, the network administrator is the one to decide whether the event corresponds to an existing cluster or whether a new cluster should be established. Our GA tool assists him by providing the level of proximity between the new event and the existing clusters. In this way, the system continuously updates itself, at the same time learning more about its environment. Thus, the main objective of this tool is to couple expert knowledge and GA data analysis technologies. An important point that should be emphasized is that our algorithm is unsupervised, i.e. it does not require training data to be labelled. In this way the extensive engineering and error-prone job of labelling network data is avoided.

In the recent past, there has been a great deal of criticism towards applying machine learning techniques in network security. The critics refer mainly to two issues:

1. Machine learning techniques do not exhibit broad applicability, i.e. they are not able to detect attacks for which they were not trained to detect
2. Most of machine learning is focused on understanding the **common** cases, but what is wanted is to find the **outliers** (that are generally not so common)

We have tried to mitigate these issues by coupling the GA with expert knowledge. In this way the GA process of learning is “revised” and moves towards the desired direction. This would not be possible without the intrinsic properties of GA, namely a high level of adaptability to the environment changes. Moreover, it is demonstrated that GAs cope well with classifying rare or **uncommon** cases. Finally, the algorithm is only one part within the broader intrusion detection [2] framework, while the critics mostly concern cases where a machine learning technique is deployed without any additional support.

The rest of the work is organized as follows. Section 2 presents related work. Section 3 details the implementation of the system. Section 4 gives the results of initial testing, while Section 5 presents novelty and advantages of the approach, draws conclusions and suggests future strategies.

2 Related Work

Most of the machine-learning techniques deployed for intrusion detection are supervised, as these techniques exhibit higher level of accuracy than the unsupervised ones. However, they have an important deficiency because they operate on labelled data. Labelling network data is time-consuming and error-prone process whose possible errors may affect negatively on the level of accuracy of deployed technique.

On the other hand, unsupervised techniques as K-means clustering, although do not suffer from the problem of data labelling, exhibit other deficiencies. The number of clusters in K-means clustering has to be determined a priori and cannot be changed during the process. This is an important deficiency, as the optimal number of clusters in an environment that is constantly changing, as network environment is, is hard to determine. Moreover, this technique is known to get stuck at sub-optimal solutions depending on the choice of the initial cluster centres. *k*NN algorithm suffers from similar problem, that consists in determining the optimal value of *k* a priori without the possibility of changing it on the fly. Again, this does not provide the level of flexibility that is necessary in dynamic environments. Furthermore, its accuracy can be severely degraded by the presence of noisy or irrelevant features, meaning that it is not robust enough.

On the other hand, GAs avoid getting stuck in local optima due to working with populations of candidate solutions rather than a single solution, and employing stochastic operators. For the same reason, they are very robust which helps them in dealing with noisy network data. Furthermore, our design does not assume that the number of clusters has to be known a priori which provides higher level of flexibility. Thus, our approach offers true prospects for gaining higher performances when dealing with network data.

3 System Implementation

The process of producing security intelligence is comprised of the following five steps depicted in Fig.1 below:

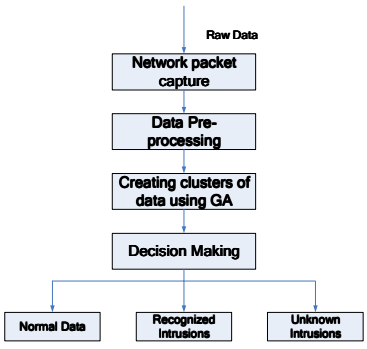

Fig. 1. Detection process flow

The algorithm whose software implementation is presented in the following uses the GAlib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology [3].

3.1 Designate Data

The input to the algorithm is raw data. The data is further captured by the Wireshark Network Protocol Analyzer [4] which provides the capture data in .csv format as output. Captured data completely defines particular connection, as it contains information about source and destination IP address, ports and payload data.

3.2 Data Pre-processing

In this step, each attribute, i.e. each feature of the pre-determined model of network connections, gets its value based upon the raw data obtained in the previous step. Thus, the output of this step consists of multi-dimensional event vectors that represent the model of network traffic. In this way every network connection is represented as a vector.

3.3 Model Training

This is the process of forming clusters of similar data. It is assumed that the number of clusters is not known *a priori*. This provides a higher level of flexibility as an unknown attack does not necessarily have to be joined to an existing cluster, yet it can form a new cluster. Initial clusters are formed in the process of training by evolving a genetic algorithm. The genetic algorithm type deployed is Incremental GA, i.e. it uses overlapping populations, but with very little overlap (only one or two individuals get replaced in each generation). The replacement scheme used is WORST, i.e. the worst-performing individuals are replaced with the new ones in each iteration [3].

The pseudocode of the training process is given in Fig.2 below. The process of training is constantly repeated at certain moments of time. Each time different dataset is used for training.

```
Set the number of individuals, the number of generations, and the
number of individuals to be changed in each generation, mutation and
crossover rate values;

Initialize the population;

For each individual from the population

            For each connection from the dataset

            Calculate fitness function according to the formula;

            Breed the population using mutation and crossover;
```

```
            Substitute the worst individuals with the new generated
ones;

            Repeat the previous steps for the specified number of
generations;
```

Fig. 2. Pseudocode of the Training Process

A further explanation of each step follows:

Individual Representation and Population Initialization. Genomes of the GA are two-dimensional matrices, where each row represents a cluster centre. GAlib provides resizable two-dimensional genomes making in this way the idea of unknown number of clusters feasible [3]. Each element of a cluster centre corresponds to a feature of the pre-determined model of network traffic. Thus, the size of a row is equal to the pre-determined number of network features that describe the network model deployed.

At the beginning of the first execution of the training process, each element of each genome of the population is initialized to a random float number. The initial population of every next training process will be the output of the previous one. The size of the population and the number of generations are set by the user.

Performance Measurement (Fitness Function). For each chromosome, the centres encoded in it are first extracted, and then a partition is obtained by assigning the points to a cluster corresponding to the closest centre. The distance between the points is computed as the Euclidian distance [5]:

$$d(X,C) = \sqrt{\sum_{i=1}^{n}(x_i - c_i)^2} \tag{1}$$

where c represents the vector of cluster centre whose dimensions are 1xn, n is the number of predetermined features used to represent the model of network traffic, and x represents the current packet converted to an event vector mentioned above. If all the distances to the corresponding centres are greater than the predefined threshold, then a new centre is established. The cluster centres encoded in a chromosome are then replaced by the centroids of the corresponding clusters. The cluster centroid is the average vector of all the vectors that belong to a cluster.

Given the above partition, and the number of clusters, the value of the cluster validity index is computed. The fitness of a chromosome is then defined as a function of the corresponding cluster validity index. The Davies-Bouldin (*DB*) index [6] is selected because of the following advantages over other measures:

1. Stability of results: this index is less sensitive to the position of a small group of data set members (so called outliers) than other measures, such as for example, the Dunn's index.
2. In the case of more than 2 clusters and the need to rank them, some measures (for example the Silhouette index) behave unpredictably, whereas the expected behaviour of the Davies-Bouldin index in these cases is good.

DB index is a function of the ratio of the sum of *within-cluster scatter* to *between-cluster separation.* The scatter within the *i*th cluster is computed as:

$$S_i = \sqrt{\frac{1}{|C_i|}\sum_{x \in C_i} \|x - c_i\|} \quad (2)$$

where C_i and $| C_i |$ represent the ith cluster number of the elements that belong to the ith cluster respectively. $\|x\text{-}c_i\|$ is the distance between the centre and an element from the same cluster. It is calculated as the Euclidean distance given in the previous formula. The distance d_{ij} between two clusters is considered to be the Euclidean distance between the corresponding cluster centres. Then, we compute:

$$R_i = \max_{j, j \neq i} \left\{ \frac{S_i + S_j}{d_{ij}} \right\} \quad (3)$$

The Davies-Bouldin DB index is then computed as:

$$DB = \frac{1}{K}\sum_{i=1}^{K} R_i \quad (4)$$

where K is the number of different clusters. The objective is to minimize the DB index for achieving proper clustering. Therefore, the fitness of chromosome j is defined as $(1/DB_j)$, where DB_j is the Davies-Bouldin index computed for this chromosome. The maximization of the fitness function will ensure minimization of the DB index.

Selection. Standard roulette-wheel selection is deployed. The possibility of selecting an individual is directly proportional to its fitness value [1].

Crossover. Standard one-point crossover is deployed. The possibility of crossover can be set by the user [1].

Mutation. A swap-mutator is deployed to carry out the process of mutation. The possibility of mutation can be set by the user [1].

The deployed selection, mutation and crossover operator are built-in the GAlib package [3].

3.4 Cluster Labelling and Decision Making

After the process of establishing the initial clusters, the next step is to classify network events. Before performing the process of classification, the clusters are labelled by the security expert. According to his previous knowledge, security expert may label the clusters as intrusive, normal or previously unseen that need further investigation. We have opted for this approach as it is the most appropriate solution for a system that is to be employed in a real-world application.

During the process of classification, the Euclidean distances between the event and the existing clusters are calculated. The event is assigned to belong to the closest cluster. However, if all the calculated distances surpass the predetermined threshold value (that determines the maximum possible distance between a network event and its corresponding cluster centre, meaning that the packet does not correspond to any of the existing clusters), the security expert is called upon to be the final arbiter. Our GA tool assists him by providing the level of proximity between the new packet and the existing clusters. The security expert renders the final decision as to whether the

event corresponds to an existing cluster or a new cluster should be established, meaning that the event has been unknown up to this moment. In this way, the system continuously updates itself, at the same time learning more about its environment.

4 Initial Testing and Results

Testing of the implemented algorithm was carried out on the benchmark KDD99 dataset [7]. As KDD set has its own pre-defined features [7], the step of data-preprocessing explained above is skipped. GA parameters of initial testing are the following ones: initial population contains 100 individuals, 50 generations are evolved during the process of evolution, with the probabilities of crossover and mutation operator 0.9 and 0.1 respectively. For the purpose of initial testing, the process of labeling is performed using the existing labels of the dataset: a cluster gets the label of the group (normal or one of four types of attacks) whose number of individuals that pertain to the group is higher than the rest.

The KDD dataset has been found to have a number of drawbacks [8], [9]. Thus the testing results do not reflect the behaviour of the algorithm in a real-world environment. Moreover, current testing is not performed in a manner the system is going to be utilized, i.e. all the tests were carried out on the initial clusters without performing any re-clustering by a security expert. The process of cluster labeling is also performed in a different way. Testing results, however, reflect a high capability that the algorithm exhibits in distinguishing intrusive connections from the non-intrusive ones. The overall detection rate of 91% with a false-positive rate of 0.5% ranks this system as one of the best-performing unsupervised methods in the current state-of-the-art. Due to the skewed distribution of different attack groups (number of DoS attacks highly surpasses the number of the rest), deployed labeling results in assigning DoS label to all of the attacks groups. Furthermore, this leads to not having alteration of the final result (expressed in the terms of detection and false-positive rate) while changing different parameters of the proposed algorithm, such as population size or possibilities of the operators, or different fitness functions (Dunn index [6] or *I*-index proposed in [10]).

The observed drawback consists in assigning some of the attacks into the clusters that represent different attacks groups, or declare them as normal. The latter is often the case with U2R (User-to-root) and R2L (Remote-to-Local) types of attacks for the dataset. This occurs partially due to the known drawback of the dataset, i.e. the dataset is proven to possess connections with absolutely the same or very similar feature values, but with different labels [8]. However, all the drawbacks are expected to be mitigated through the influence of the security expert.

5 Conclusions and Future Strategies

The described GA design represents a novelty in the network security field. Most of the existing applications of GAs for intrusion detection assume supervised learning where the training dataset needs to be labelled. Our design of the algorithm is

unsupervised, so we have avoided the process of labelling the data which can be time-consuming and error-prone. On the other hand, various clustering algorithms have been deployed for classifying network events, but none of them was based on genetic algorithms. Hence, the algorithm as deployed presents a novelty in the network security field. Moreover, our system can cooperate with a security expert providing in that way higher accuracy and adaptability to environmental changes.

Initial testing reflects a high capability that the algorithm exhibits in distinguishing intrusive connections from the non-intrusive ones. In the near future, the algorithm will be tested in a real-world environment as a part of the final product of the Intrusion Detection Project [2].

Strategies for the continuation and the improvement of the work presented here are numerous and concern various aspects of the algorithm. The question of data pre-processing still remains unresolved. A promising solution could be to deploy the idea presented in PAYL [11]. More flexibility could be given to the algorithm by providing different options for performance measurement, selection, crossover and mutation. Moreover, there is a great possibility of using different standard genetic operators for selection (rank, tournament, deterministic sampling) or crossover (two-point, even-odd) and mutation (random flip) or even developing custom operators [3], [1].

Acknowledgements. This work has been partially funded by the Spanish Ministry of Education and Science under the project TEC2006-13067-C03-03 and by the European Commission under the FastMatch project FP6 IST 27095.

References

1. Goldberg D: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley Longman, Inc. 1989
2. www.fastmatch.org
3. GAlib A C++ Library of Genetic Algorithm Components, http://lancet.mit.edu/ga/
4. www.wireshark.org, accessed during 2007
5. Richard O. Duda, Peter E. Hart, David G .Stork: Pattern Classification, 2nd Edition, Wiley InterScience, October 2000
6. Bolshakova N. and Azuaje F.: Cluster Validation Techniques for Genome Expression Data, Signal Processing, 83, 2003, pp. 825-833.
7. KDD Cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, October 1999, accessed during 2006 and 2007
8. Bouzida Y and Cuppens F, : Detecting Novel and Known Intrusions, IFIP/SEC 2006, 21st IFIP TC-11 International Information Security Conference Karlstad University, Karlstad, Sweden. May 2006.
9. McHugh J, :Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Library, ACM Transactions on Information and System Security, Vol. 3, No. 4, pp. 262-294, November 2000
10. Bandyopadhyay, S. and Maulik, U.: Nonparametric genetic clustering: comparison of validity indices, IEEE Transactions on Systems, Man, Cybernetics, Part C, 2001
11. Ke Wang and Salvatore J. Stolfo. : Anomalous Payload-based Network Intrusion Detection. RAID, Sept., 2004.