GP Ensemble for Distributed Intrusion Detection Systems

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano

ICAR-CNR,
Via P.Bucci 41/C,
Univ. della Calabria
87036 Rende (CS), Italy
{folino,pizzuti,spezzano}@icar.cnr.it

Abstract. In this paper an intrusion detection algorithm based on GP ensembles is proposed. The algorithm runs on a distributed hybrid multi-island model-based environment to monitor security-related activity within a network. Each island contains a cellular genetic program whose aim is to generate a decision-tree predictor, trained on the local data stored in the node. Every genetic program operates cooperatively, yet independently by the others, by taking advantage of the cellular model to exchange the outmost individuals of the population. After the classifiers are computed, they are collected to form the GP ensemble. Experiments on the KDD Cup 1999 Data show the validity of the approach.

1 Introduction

The extensive use of Internet and computer networks, besides the known advantages of information exchange, has provided also an augmented risk of disruption of data contained in information systems. In the recent past, several cyber attacks have corrupted data of many organizations creating them serious problems. The availability of *Intrusion Detection Systems(IDS)*, able to automatically scan network activity and to recognize intrusion attacks, is very important to protect computers against such unauthorized uses and make them secure and resistant to intruders. The task of an IDS is, in fact, to identify computing or network activity that is malicious or unauthorized. Most current Intrusion Detection Systems collect data from distributed nodes in the network and then analyze them centrally. The first problem of this approach is a security and privacy problem due to the necessity of transferring data. Moreover, if the central server becomes the objective of an attack, the whole network security is quickly compromised.

In this paper an intrusion detection algorithm ($GEdIDS$, *Genetic programming Ensemble for Distributed Intrusion Detection Systems*), based on the ensemble paradigm, that employs a Genetic Programming-based classifier as component learner for a distributed Intrusion Detection System ($dIDS$), is proposed.

To run genetic programs a distributed environment, based on a hybrid multi-island model [2] that combines the island model with the cellular model, is used. Each node of the network is considered as an island that contains a learning

component, based on cellular genetic programming, whose aim is to generate a decision-tree predictor trained on the local data stored in the node. Every genetic program, however, though isolated, cooperates with the neighboring nodes by collaborating with the other learning components located on the network and by taking advantage of the cellular model to exchange the outmost individuals of the population.

A learning component employs the ensemble method AdaBoost.M2 [7] thus it evolves a population of individuals for a number of rounds, where a round is a fixed number of generations. Every round the islands import the remote classifiers from the other islands and combine them with their own local classifier. Finally, once the classifiers are computed, they are collected to form the GP ensemble. In the distributed architecture proposed, each island thus operates cooperatively yet independently by the others, providing for efficiency and distribution of resources. This architecture gives significant advantages in scalability, flexibility, and extensibility. To evaluate the system proposed, experiments using the network records of the KDD Cup 1999 Data [1] have been performed. Experiments on this data set point out the capability of genetic programming in successfully dealing with the problem of distributed intrusion detection.

Genetic Programming for intrusion detection has not been explored very much. Some proposals can be found in [11, 9, 8]

The paper is organized as follows. The next section introduces the genetic programming based ensemble paradigm for distributed IDS and describes the distributed programming environment proposed. Section 3 presents the results of experiments.

2 GP ensembles for dIDS

Ensemble is a learning paradigm where multiple component learners, also called classifiers or predictors, are trained for a same task by a learning algorithm, and then combined together for dealing with new unseen instances. Ensemble techniques have been shown to be more accurate than component learners constituting the ensemble [4, 10], thus such a paradigm has become a hot topic in recent years and has already been successfully applied in many application fields.

In this paper such a paradigm has been adopted for modelling distributed intrusion detection systems and the suitability of genetic programming (GP) as component learner has been investigated. The approach is based on the use of cooperative GP-based learning programs that compute intrusion detection models over data stored locally at a site, and then integrate them by applying a majority voting algorithm. The models are built by using the local audit data generated on each node by, for example, operating systems, applications, or network devices so that each ensemble member is trained on a different training set.

The GP classifiers cooperate using a multi-island model to produce the ensemble members. Each node is an island and contains a GP-based learning component extended with the boosting algorithm AdaBoost.M2 [7] whose task is to

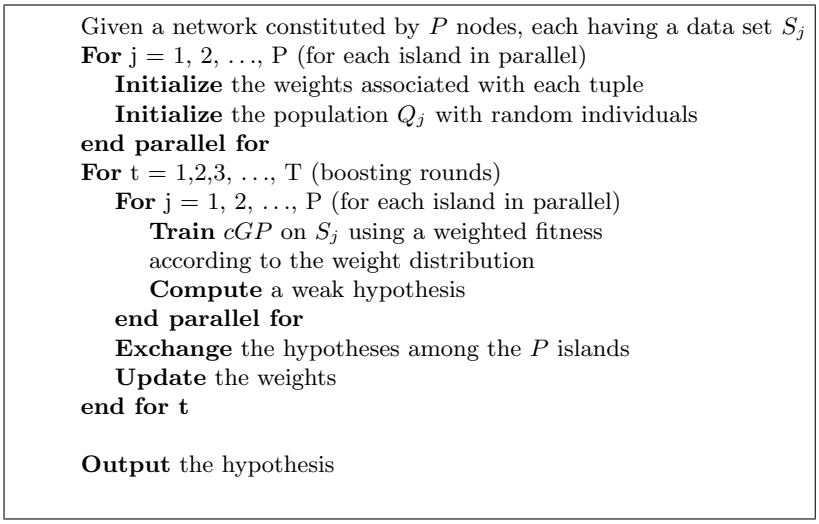

Fig. 1. The GEdIDS algorithm using AdaBoost.M2

build a decision tree classifier by collaborating with the other learning components located on the network. Each learning component evolves its population for a fixed number of iterations and computes its classifier by operating on the local data. Each island may then import (remote) classifiers from the other islands and combine them with its own local classifiers to form the GP ensemble.

In order to run GP ensembles a distributed computing environment is required. We use dCAGE (distributed Cellular Genetic Programming System) a distributed environment to run genetic programs by a multi-island model, which is an extension of [6]. An hybrid variation of the classic multi-island model, that combines the island model with the cellular model, has been implemented.

The Island model is based on subpopulations, that are created by dividing the original population into disjunctive subsets of individuals, usually of the same size. Each subpopulation can be assigned to one processor and a standard (panmictic) GP algorithm is executed on it. Occasionally, migration process between subpopulations is carried out after a fixed number of generations. The hybrid model modifies the island model by substituting the standard GP algorithm with a cellular GP (cGP) algorithm [6]. In the cellular model each individual has a spatial location, a small neighborhood and interacts only within its neighborhood. The main difference in a cellular GP, with respect to a panmictic algorithm, is its decentralized selection mechanism and the genetic operators (crossover, mutation) adopted.

In dCAGE, to take advantage of the cellular model of GP, the cellular islands are evolved independently, and the outmost individuals are asynchronously exchanged so that all islands can be thought as portions of a single population. dCAGE distributes the evolutionary processes (islands) that implement the de-

tection models over the network nodes using a configuration file that contains the configuration of the distributed system. dCAGE implements the hybrid model as a collection of cooperative autonomous islands running on the various hosts within an heterogeneous network that works as a peer-to-peer system. Each island, employed as a peer IDS, is identical to each other except one that is supposed to be the *collector* island. The collector island is in charge of collecting the GP classifiers from the other nodes, handling the fusion of results on behalf of the other peer IDS, and to redistribute the GP ensemble for future predictions to all network nodes. Respect to the other islands, it has some more duties for administration. The configuration of the structure of the processors is based on a ring topology.

The pseudo-code of the algorithm is shown in figure 1. Each island is furnished with a cGP algorithm enhanced with the boosting technique AdaBoost.M2, a population initialized with random individuals, and operates on the local audit data weighted according to a uniform distribution. The selection rule, the replacement rule and the asynchronous migration strategy are specified in the cGP algorithm. Each peer island generates the GP classifier iterating for a certain number of iterations necessary to compute the number of boosting rounds. During the boosting rounds, each classifier maintains the local vector of the weights that directly reflect the prediction accuracy on that site. At each boosting round the hypotheses generated by each classifier are stored and combined in the collector island to produce the ensemble of predictors. Then the ensemble is broadcasted to each island to locally recalculate the new vector of the weights. After the execution of the fixed number of boosting rounds, the classifiers are used to evaluate the accuracy of the classification algorithm for intrusion detection on the test set.

Genetic programming is used to inductively generate a GP classifier as a decision trees for the task of data classification. Decision trees, in fact, can be interpreted as composition of functions where the function set is the set of attribute tests and the terminal set are the classes. The function set can be obtained by converting each attribute into an attribute-test function. For each attribute A, if $A_1, \ldots A_n$ are the possible values A can assume, the corresponding attribute-test function f_A has arity n and if the value of A is A_i then $f_A(A_1, \ldots A_n) = A_i$. When a tuple has to be evaluated, the function at the root of the tree tests the corresponding attribute and then executes the argument that outcomes from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise the new function is executed. The fitness is the number of training examples classified in the correct class. The *Cellular genetic programming* algorithm (*cGP*) for data classification has been proposed in [5] and it is described in figure 2. At the beginning, for each cell, the fitness of each individual is evaluated. Then, at each generation, every tree undergoes one of the genetic operators (reproduction, crossover, mutation) depending on the probability test. If crossover is applied, the mate of the current individual is selected as the neighbor having the best fitness, and the offspring is generated. The current tree is then replaced by the best of the two offspring if the fitness of the latter

```
Let p_c, p_m be crossover and mutation probability
for each point i in the population do in parallel
   evaluate the fitness of t_i
end parallel for
while not MaxNumberOfGeneration do
   for each point i in the population do in parallel
      generate a random probability p
      if (p < p_c)
         select the cell j, in the neighborhood of i,
         such that t_j has the best fitness
         produce the offspring by crossing t_i and t_j
         evaluate the fitness of the offspring
         replace t_i with the best of the two offspring
         if its fitness is better than that of t_i
      else
      if ( p < p_m + p_c) then
         mutate the individual
         evaluate the fitness of the new t_i
      else
         copy the current individual in the population
      end if
      end if
end parallel for
end while
```

Fig. 2. The algorithm cGP

is better than that of the former. The evaluation of the fitness of each classifier is calculated on the entire training data. After the execution of the number of generations defined by the user, the individual with the best fitness represents the classifier.

3 System evaluation and results

3.1 Data sets description

Experiments over the KDD Cup 1999 Data set [1] have been performed. This data set comes from the 1998 DARPA Intrusion Detection Evaluation Data and contains a training data consisting of 7 weeks of network-based intrusions inserted in the normal data, and 2 weeks of network-based intrusions and normal data for a total of 4,999,000 connection records described by 41 characteristics. The main categories of intrusions are four: Dos (Denial Of Service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to a local superuser privileges by a local unprivileged user), PROBING (surveillance and probing). However a smaller data set consisting of the 10% the overall data set

is generally used to evaluate algorithm performance. In this case the training set consists of 494,020 records among which 97,277 are normal connection records, while the test set contains 311,029 records among which 60,593 are normal connection records. Table 1 shows the distribution of each intrusion type in the training and the test set.

Table 1. Class distribution for training and test data for KDDCUP 99 dataset

	Normal	**Probe**	**DoS**	**U2R**	**R2L**	**Total**
Train	97277	4107	391458	52	1126	494020
Test	60593	4166	229853	228	16189	311029

3.2 Performance measures

To evaluate our system, besides the classical accuracy measure, the two standard metrics of *detection rate* and *false positive rate* developed for network intrusions, have been used. Table 2 shows these standard metrics. Detection rate is computed as the ratio between the number of correctly detected intrusions and the total number of intrusions, that is $DR = \frac{\#TruePositive}{\#FalseNegative+\#TruePositive}$. False positive (also said false alarm) rate is computed as the ratio between the number of normal connections that are incorrectly classifies as intrusions and the total number of normal connections, that is $FP = \frac{\#FalsePositive}{\#TrueNegative+\#FalsePositive}$. These metrics are important because they measure the percentage of intrusions the system is able to detect and how many misclassifications it makes. To visualize the trade-off between the false positive and the detection rates, the ROC (Receiving Operating Characteristic) curves are also depicted. Furthermore, to compare classifiers it is common to compute the area under the ROC curve, denoted as AUC [3]. The higher the area, the better is the average performance of the classifier.

Table 2. Standard metrics to evaluate intrusions.

		Predicted label	
		Normal	Intrusions
Actual Class	Normal	True Negative	False Positive
	Intrusions	False Negative	True Positive

3.3 Experimental setup

The experiments were performed by assuming a network composed by 10 dual-processor 1,133 Ghz Pentium III nodes having 2 Gbytes of memory. Both the training set of 499,467 tuples and the test set of 311029 tuples have been equally partitioned among the 10 nodes by picking them at random. Each node thus contains 1/10 of the instances for each class. On each node we run $AdaBoost.M2$ as base GP classifier with a population of 100 elements for 10 rounds, each round consisting of 100 generations. The GP parameters used where the same for each node and they are shown in table 3. All the experiments have been obtained by running the algorithm 10 times and by averaging the results. Each ensemble has been trained on the train set and then evaluated on the test set.

Table 3. Main parameters used in the experiments

Name	Value
max_depth_for_new_trees	6
max_depth_after_crossover	17
max_mutant_depth	2
grow_method	RAMPED
selection_method	GROW
crossover_func_pt_fraction	0.7
crossover_any_pt_fraction	0.1
fitness_prop_repro_fraction	0.1
parsimony_factor	0

3.4 Results and comparison with other approaches

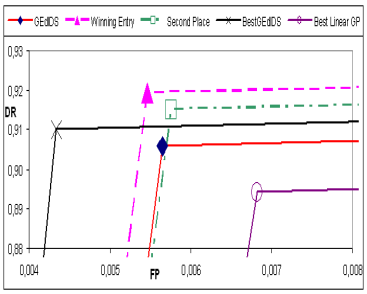

Fig. 3. ROC curves.

Table 4. Confusion matrix (averaged over all tries) for dCage. Rows show the true class, columns the predicted ones.

	Normal	**Probe**	**DoS**	**U2R**	**R2L**
Normal	60250.8	200.2	110.8	15.4	15.8
Probe	832.8	2998.4	263.6	26.4	44.8
DoS	7464.2	465.0	221874.8	19.2	29.8
U2R	139.6	45.2	17.2	11.8	14.2
R2L	15151.4	48.6	232.4	173.8	582.8

Table 5. Comparison with kdd-99 cup winners and other approaches

Algorithm	**Detection Rate**	**FP Rate**	**ROC Area**
Winning Entry	0.919445	0.005462	0,956991
Second Place	0.915252	0.005760	0,954746
Best Linear GP - FP Rate	0.894096	0.006818	0,943639
Avg GEdIDS	0.905812	0.005648	0.950082
Best GEdIDS - FP Rate	0.910165	0.004340	0.952912

The results of our experiments are summarized in table 4, where the confusion matrix obtained on the test set by averaging the 10 confusion matrices coming from 10 different executions of $GEdIDS$ is showed. The table points out that the prediction is worse on the two classes U2R and R2L. For this two classes, however, there is a discrepancy between the number of instances used to train each classifier on every node and the number of instances to classify in the test set. Table 5 compares our approach with the first and second winner of the KDD-99 CUP competition and the linear genetic programming approach proposed by Song et al. [11]. The table shows the values of the standard metrics described above. In particular we show the detection rate, the false positive rate, and the ROC area of these three approaches and those obtained by $GEdIDS$. For the latter we show both the average values of the 10 executions and the best value with respect to the false positive rate. From the table we can observe that the performance of $GEdIDS$ is comparable with that of the two winning entries and better than Linear GP. In fact, the average and best $GEdIDS$ detection rates are 0.905812 and 0.910165, respectively, while those of the first two winners are 0.919445 and 0.915252. As regard the false positive rate the average value of $GEdIDS$ 0.005648 is lower than the second entry, while the best value obtained 0.004340 is lower than both the first and second entries. Thus the solutions found by $GEdIDS$ are very near to the winning entries, and, in any case, overcome those obtained with linear GP. These experiments emphasizes the capability of genetic programming to deal with this kind of problem. Finally figure 3 shows an enlargement of the ROC curves of the methods listed in table 5 and better highlights the results of our approach.

4 Conclusions

A distributed intrusion detection algorithm based on the ensemble paradigm has been proposed and the suitability of genetic programming as a component learner of the ensemble has been investigated. Experimental results show the applicability of the approach for this kind of problems. Future research aims at extending the method when considering not batch data sets but data streams that change online on each node of the network.

Acknowledgments

This work has been partially supported by projects CNR/MIUR legge 449/97-DM 30/10/2000 and FIRB Grid.it (RBNE01KNFP).

References

1. The third international knowledge discovery and data mining tools competition dataset kdd99-cup. In *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*, 1999.
2. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transaction on Evolutionary Computation*, 6(5):443–462, October 2002.
3. A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.
4. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
5. G. Folino, C. Pizzuti, and G. Spezzano. A cellular genetic programming approach to classification. In *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, pages 1015–1020, Orlando, Florida, July 1999. Morgan Kaufmann.
6. G. Folino, C. Pizzuti, and G. Spezzano. A scalable cellular implementation of parallel genetic programming. *IEEE Transaction on Evolutionary Computation*, 7(1):37–53, February 2003.
7. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th Int. Conference on Machine Learning*, pages 148–156, 1996.
8. Wei Lu and Issa Traore. Detecting new forms of network intrusion using genetic programming. In *Proc. of the Congress on Evolutionary Computation CEC'2003*, pages 2165–2173. IEEE Press, 2003.
9. Srinivas Mukkamala, Andrew H. Sung, and Ajith Abraham. Modeling intrusion detection systems using linear genetic programming approach. In *17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004*, pages 633–642, Ottawa, Canada, 2004.
10. J. Ross Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence AAAI96*, pages 725–730. Mit Press, 1996.
11. D. Song, M.I. Heywood, and A. Nur Zincir-Heywood. A linear genetic programming approach to intrusion detection. In *Proceedings Of the Genetic and Evolutionary Computation Conference GECCO 2003*, pages 2325–2336. LNCS 2724, Springer, 2003.