2010 International Conference on Availability, Reliability and Security

Improving Network Intrusion Detection by Means of Domain-Aware Genetic Programming

Jorge Blasco*, Agustin Orfila†, Arturo Ribagorda‡
Computer Science Department
Carlos III University of Madrid
Leganés, Spain
{*jbalis**, *adiaz*†, *arturo*‡}@*inf.uc3m.es*

***Abstract*—One of the central areas in network intrusion detection is how to build effective systems that are able to distinguish normal from intrusive traffic. In this paper we explore the use of Genetic Programming (GP) for such a purpose. Although GP has already been studied for this task, the inner features of network intrusion detection have been systematically ignored. To avoid the blind use of GP shown in previous research, we guide the search by means of a fitness function based on recent advances on IDS evaluation. For the experimental work we use a well-known dataset (i.e. KDD-99) that has become a standard to compare research although its drawbacks. Results clearly show that an intelligent use of GP achieves systems that are comparable (and even better in realistic conditions) to top state-of-the-art proposals in terms of effectiveness, improving them in efficiency and simplicity.**

***Keywords*-intrusion detection; genetic programming; efficiency; effectiveness;**

I. Introduction

Intrusion Detection is the process of monitoring and analyzing the activity of a network or a computer system in order to detect possible intrusion attacks [1]. The design of a network intrusion detection system (NIDS) is determined by a set of decisions about raw data obtaining, event detection, analysis rules, data storage and response procedures. Focusing on the analysis techniques, artificial intelligence has been widely explored, including approaches based on machine learning, neural networks, evolutionary computation, etc.

In this paper we focus on the improvement of automatic generation of analysis rules using Genetic Programming (GP). Our research tries to improve the results on effectiveness found in the literature while enhancing the efficiency and semantics of the solutions. Thus, in terms of effectiveness, the way we approach to GP provides IDS analysis rules that at least achieve the same level of state-of-the-art proposals. In addition, our system clearly outperforms classical machine learning algorithms when the dataset is adapted to have a more realistic prevalence of attacks. For a NIDS is not only important the effectiveness but also the efficiency. In intensive network usage environments IDS must analyze huge amounts of data. If the NIDS is not fast enough it will begin to drop the analysis of packets. In this regard, the solutions provided by algorithms like C4.5 [2] generate wide and deep trees which may produce an overhead on the analysis process. On the contrary, GP trees can be quite simple being able to process more information in less time. Furthermore, the use of an appropriate function set for GP individuals results in analysis rules that provide better knowledge about the nature of the attacks. Other paradigms involve specialized structures which are nothing like computer programs (e.g. weight vectors for neural networks) what constrains the semantics of the generated rules.

In addition to the use of traditional metrics to evaluate our IDS, we have also used a recently presented metric (i.e. C_{id}) [3] proposed specifically for the intrusion detection domain. Our recent research [4] has proved that domain-aware GP is able to produce efficient and easy to understand rules for IDS, specifically to detect probe attacks. In our efforts to provide an exhaustive comparison of the efficiency of our approach it was necessary to use a dataset which covered a wide range of different attacks types. To evaluate our approach we have used the well known KDD-99 dataset. Although this dataset has been criticized in some studies [5], [6] due to questions such as its unrealistic prevalence of attacks or its uncertain relation with reality, it is still used in recent publications [7] and is considered as a standard benchmark that most research uses to measure effectiveness.

The remainder of this paper is structured as follows. Section II briefly reviews the basics of genetic programming. Section III reviews related work done in the area. Section IV describes the design of the proposed system. Section V describes de KDD-99 Dataset. Then, Section VI shows the experimental setup, results and discussion. Finally, last section summarizes the main conclusions and future work.

II. Genetic Programming Basics

Genetic Programming is a supervised search technique devised by John R. Koza in 1992 [8]. GP is somehow similar to Genetic Algorithms (GA), but instead of using chromosomes to encode the solution, it uses computer programs represented as trees. IDS are itself computer programs and its size and structure is not known in advance. Consequently, the use of GP is more appropriate than GA for the problem at hand. The design of a GP algorithm requires the definition of these elements:

- A population of individuals. The initial population often consists of randomly generated individuals. Each individual codifies a computer program or mathematical function. It is usually represented by a tree composed of functions and terminal. The function set specifies which kind of functions can be part of the individuals. The terminal set comprises all possible parameters for the functions. Ephemeral Random Constants (ERCs) are a highly used kind of terminals. An ERC is a terminal node of the tree which is initialized randomly and returns always the same value.
- A fitness function. For each generation, every individual of the population must pass through the process of natural selection. The fitness function evaluates the quality of each individual.
- A set of genetic operators. For each generation of a GP algorithm, some operations are performed on the population. These operations are reproduction, mutation and crossover. Reproduction does not change the individual but generates an offspring from a given population. Mutation randomly changes a function, a terminal or a complete subtree of an individual. Finally, crossover performs exchanges on two subtrees from two individuals, thereby combining characteristics from both of them into the new offspring.

A basic GP algorithm consists of a number of cycles. At each cycle the fitness function is evaluated over the population and the genetic operators are subsequently applied, thus producing consecutive generations of populations of computer programs, until an ending condition is reached.A typical GP implementation has various parameters to be adjusted, such as the population size and the maximum number of generations.

III. Related Work

The use of GP to generate IDS analysis rules was first proposed in [9]. Standard GP as described by Koza has not been often used in the IDS domain. Instead, use of different variants of standard GP has been proposed, namely Linear Genetic Programming (LGP), Multi Expression Programming (MEP) and Gene Expression Programming (GEP) [10]. LGP evolves individuals described in an imperative programming language like C [11]. In LGP, the minimum unit of evolution is a native machine code instruction. Song et al. [12] applied LGP to the IDS domain obtaining similar, but worse results than KDD-99 winner [13]. They also claimed that solutions provided by their GP algorithm could be used to extract knowledge. Nevertheless, their knowledge extraction is focused on the results over the test dataset and not on the inner characteristics of the GP individual. In MEP each individual encodes several expressions. The best expression of the individual is chosen as representative for the individual. In GEP, individuals are encoded as linear chromosomes which are expressed as expression trees. The genotype is defined by the linear chromosomes and the phenotype is defined by expression trees.

Abraham et al. [14], [15] reviewed and compared these three derivates of standard GP and applied them to IDS analysis rules generation. Results, which look good, were obtained using training and testing sets which were a small subset of the KDD-99 dataset. Therefore, results obtained were not computed from the complete original dataset but from an ad-hoc subset (not publicly available). This situation makes a comparison with state-of-the-art unfeasible. As a consequence, our experimental work focuses on the original KDD-99 dataset in order to achieve a fair comparison with top state-of-the-art. Faraoun [16] also proposed the use of GP to evolve a multi-category classifier IDS. Results obtained on the KDD-99 dataset were good, but the chosen function set was composed by non-linear functions. Besides the lack of justification for this function selection, these functions do not allow the understanding of the generated individuals, losing one of the potentials of the GP approach.

Many other approaches have been used to find good analysis rules for IDS. Unsupervised techniques such as cluster-based classification, K-means and SVM were proposed on [17]. On section VI these works are compared to our proposal in terms of effectiveness, efficiency and semantics.

IV. Design

The aim of this work is to show how domain-aware GP can help in the production of effective, efficient and easy to understand analysis rules to differentiate normal network traffic from intrusive one. The design of the GP algorithm requires the definition of the elements described in section II such as the population of individuals, the function set, the terminal set and the fitness function.

A. Population of individuals

Each individual of the population represents a potential analysis rule for an IDS. An analysis rule decides (depending on its input) whether there is an intrusion attempt or not. The inputs of our analysis rules are a set of features describing a connection and the network state (in our evaluation a KDD-99 dataset entry).

B. Function Set Selection

One of the goals of using GP in the intrusion detection field is to produce analysis rules which can be used to extract domain knowledge. Functions have been selected having this idea in mind. As the algorithm is designed to distinguish between two categories (i.e. normal and intrusive), our functions produce only two possible values as output. Thus, our functions are modified versions of the typical logic operators *and*, *or*, *not*, *greater*, *least*, *equal* and *different* that produce only two logical values (i.e. 0 or 1).

C. Terminal Set Selection

First group of terminals consists of each feature of KDD-99 records (a total of 41). Second is composed of two Ephemeral Random Constants (ERC). The first ERC takes real values between -1.0 and 1.0. The second takes integer values and is constrained between 0 and 100. This will allow the GP algorithm to generate rules such as "$port_i == 80$" which will allow easy extraction of knowledge.

D. Fitness Function

Usually the prevalence of attacks is very low in a real network. As a consequence, accuracy (i.e. percentage of hits) is not appropriate to measure the effectiveness of an IDS. In this regard, it is important to note that a naive IDS that states that every event is intrusive would achieve good accuracy being a bad detector. Nevertheless, we have observed that most GP approaches to intrusion detection rules have used a fitness function based on accuracy [16], [15]. To avoid this kind of situations, classical IDS effectiveness evaluation provides the trade-off between the hit rate (H) and the false alarm rate (F) (defined in the next subsection). Recently a new one-dimensional metric, namely C_{id} [3], has been proposed. It is based on information theory and takes into account the hit rate, the false alarm rate and the prevalence of attacks. Consequently, we have used two different fitness functions. The former is a basic metric on the IDS domain which measures the difference between the hit and the false alarm rate. The latter is the Intrusion Detection Capability (C_{id}). In the following we describe both.

1) Hit Rate Minus False Alarm Rate: We define the hit rate (H) and the false alarm rate (F) as follows:

$$H = \frac{Attacks\ classified\ as\ Attacks}{Attacks} \tag{1}$$

$$F = \frac{Non-attacks\ classified\ as\ Attacks}{Non-attacks} \tag{2}$$

H and F provide a two-dimensional measure of the quality of individuals. Our first fitness function transforms this two-dimensional space in a one dimensional measure which aims to maximize H while minimizing F:

$$Fit_1 = H - F \tag{3}$$

Therefore, an individual is better as it has higher H and lower F. The main drawback of this function is that two very different individuals can have the same fitness. For instance, let us suppose two individuals with $Fitness_1 = 0.8$. One can have $H = 0.8$ and $F = 0$ and the other $H = 1$ and $F = 0.2$. Although this drawback, we have used this fitness function to provide an exhaustive comparison against other proposals. The second fitness tries to overcome the aforementioned drawback.

2) Intrusion Detection Capability: Our second fitness function is derived from recent research on IDS evaluation. Recent studies have proposed a new metric for IDS effectiveness evaluation called Intrusion Detection Capability (C_{id}) which is based on information theory [3]. C_{id} measures the amount of uncertainty of the input resolved once the IDS output is obtained, producing one single value for measuring IDS effectiveness. Unlike other IDS metrics, like our first fitness function, C_{id} takes into account the prevalence of attacks in the dataset (B) besides the hit rate (H) and the false positive rate (F):

$$\begin{aligned} Fit_2 = C_{id} = & - BHlog\frac{BH}{BH+HF} \\ & - B(1-H)log\frac{B(1-H)}{B(1-H)+(1-B)(1-F)} \\ & - (1-B)(1-F)log\frac{(1-B)(1-F)}{(1-B)(1-F)+B(1-H)} \\ & - (1-B)Flog\frac{(1-B)F}{(1-B)F+BH} \end{aligned} \tag{4}$$

C_{id} is more sensitive than traditional metrics in realistic situations. Authors proved that in real scenarios, in which the percentage of intrusions in the whole traffic data is low, small changes on F have a high impact on C_{id}. More details on this metric can be found on [3], [18].

V. The KDD-99 Dataset

As stated before, we have used the KDD-99 dataset [19] that derives from raw traffic captured during MIT/LL 1998 evaluation [20]. This dataset was first employed for a machine learning competition over the intrusion detection domain. Training and testing datasets were created at Columbia University after a data mining process on the raw MIT/LL data. Raw data was divided in connection records that covered about 100 bytes of a connection between two parties in a limited time. Each connection record is described by 41 attributes and also contains the corresponding class indicating if it presents normal or hostile activity. A complete description of the each attribute can be found in [21]. The dataset is available at California University website[1].

Table I shows the distribution of attacks and normal traffic instances on KDD-99. Unfortunately, the KDD dataset does not represent at all a realistic prevalence of attacks. This has been one of the main critics to this dataset [5], [6]. To avoid this drawback, some researchers have proposed the modification of the prevalence of attacks in the dataset. We have tested our approach against both scenarios in order to achieve, first, a quantitative and fair comparison with state-of-the-art and, second, a measure of effectiveness under a realistic framework. For the first set of experiments we have used the original dataset. For the second, each normal traffic record has been replicated 4.000 times, as in [18], providing a more realistic prevalence of attacks (i. e. 0.001).

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Table I
CLASS DISTRIBUTION OF THE KDD-99 DATASET

	Train	Test
Normal	97278(19.69%)	60593(19.48%)
Intrusions	396743(80.31%)	250428(80.52%)
Total	494021(100%)	311021(100%)

VI. EVALUATION AND RESULTS

We have evaluated our approach through an execution of several experiments considering the fitness functions defined in Section IV-D. To perform our experiments we have used ECJ[2], which is an evolutionary computation framework developed in Java.

A. Experimental Setup

An experiment consists on evolving individuals on the training data and then testing the best individuals over the testing dataset. Experiments have been carried out using both fitness functions. Our set of experiments has been divided in two groups. First uses the original KDD-99 to perform training and testing. Second uses the modified version of the dataset in order to test a more realistic scenario. In this case, experiments have been carried out only with the C_{id} fitness function, as the modifications on the KDD-99 do not affect H nor F. Each experiment has been repeated 30 times with different randomly generated seeds. Overall, three different trainings have been performed (Table II).

Table II
DESCRIPTION OF EXPERIMENTS PERFORMED

	Fitness Function	Dataset	Runs
Experiment 1	$H-F$	Original KDD-99	30
Experiment 2	C_{id}	Original KDD-99	30
Experiment 3	C_{id}	Modified KDD-99	30

Every experiment has been performed using the same parameters (Table III). For each experiment, maximum depth of generated trees has been restricted in order to obtain more efficient and simpler individuals which allow an easy interpretation from generated rules.

B. Results

The evolution of the best individual through generations over the training dataset is depicted in Figure 1. Although experiments correspond to different fitness functions they have been all represented in a single plot for comparison purposes. The improvement that the evolutive process produces is greater for the second fitness.

As depicted on Figure 2, increasing the number of generations in any experiment would not produce any significant improvement on the GP individuals.

[2]http://cs.gmu.edu/ eclab/projects/ecj/

Table III
LIST OF PARAMETERS USED IN EXPERIMENTS DESCRIBED IN TABLE II

Parameter	Value
Generations	52
Individuals	1024
Maximum depth	6 and 4
Initialization	Ramped Half and Half
Mutation Probability	0.1
Crossover Probability	0.9
Elitisim	Yes
Selection Mechanism	Tournament
Tournament Size	7

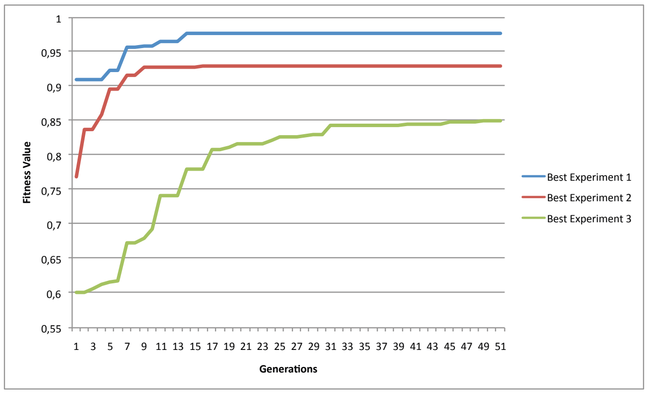

Figure 1. Fitness evolution for the best individual in each experiment

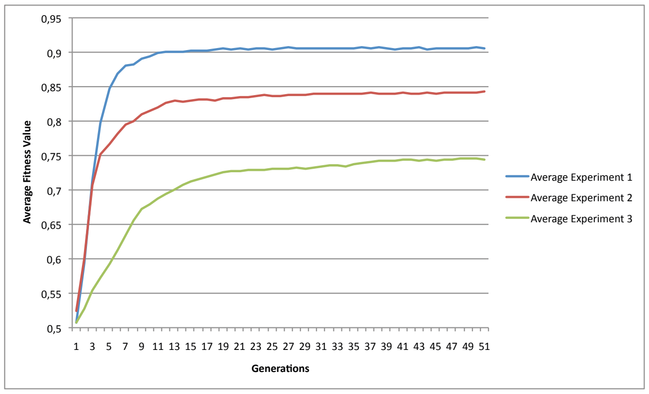

Figure 2. Average fitness evolution in each experiment during the training process.

Table IV compares the best and average results obtained in experiments 1 and 2 with top state-of-the-art proposals that refer results to the original test dataset. There are other results in the literature that even overcome the KDD-99 Winner, but they have been obtained using filtered versions (which are no publicly available) of the KDD-99 dataset. In order to have a fair comparison we have not included them. The best individual obtained in experiment 1 achieves similar results to the leading system [13]. In addition, the

Table IV
COMPARISON OF BEST INDIVIDUALS AGAINST STATE-OF-THE-ART RESULTS USING THE ORIGINAL DATASET

	H	F	C_{id}
KDD Winner [13]	0.9181	**0.0055**	**0.6774**
Best Linear GP [12]	0.8941	0.0068	0.6226
Faraoun GP [16]	0.925	0.0135	0.6630
Support Vector Machines [17]	**0.98**	0.10	0.5534
Self Organizing Maps [22]	0.89	0.046	0.4894
C 4.5 Tree	0.9099	0.0056	0.6590
Best Experiment 1	0.9131	0.0133	0.6565
Best Experiment 2	0.9180	0.0143	0.6652
Average Experiment 1	0.8024	0.0621	0.3988
Average Experiment 2	0.7915	0.0579	0.5006

best individual obtained in experiment 2 offers a better C_{id} than the one from experiment 1 and is very close to the KDD winner, beating all other approaches. These results show the suitability of the GP design proposed, particularly the use of C_{id} as fitness function.

Results achieved by the best individual of experiment 3 demonstrate the high sensitivity of C_{id} metric for a realistic prevalence of attacks. Thus, as summarized in Table V, it clearly outperforms classical machine learning algorithms in terms of C_{id} being a 15% more effective than SVM. The best individual achieves an extremely low ratio of false positives while keeping a hit rate close to the best.

Table V
COMPARISON OF BEST INDIVIDUALS AGAINST STATE-OF-THE-ART RESULTS USING THE MODIFIED DATASET

	H	F	C_{id}
Naive Bayes [18]	0.9002	0.0257	0.4258
C4.5 Tree [18]	**0.9103**	0.0176	0.4258
Support Vector Machines [18]	0.8761	0.0036	0.5642
Best Experiment 3	0.8938	**8.747E-4**	**0.7105**
Average Experiment 3	0.7626	0.0069	0.4994

Table VI[3] shows the best individuals obtained represented as C function calls. Shorter individuals are easier to interpret. We have obtained one extremely short (best experiment 1) with high effectiveness (in terms of H, F and C_{id}).

As an example, Figure 3 represents best individual obtained in experiment 1[4].This analysis rule states (besides more implications) that if there are more than one connection to the same host ($attr32$) and at least one of them is from the same service ($attr35$), but in the current record there is no data transmitted ($attr6$) because there was an error ($attr4$), then there is an intrusion going on.

Programs generated by means of GP can be composed of a relatively small set of function calls (simple operators

[3]$attr_i$ refers to feature number i of the KDD-99 dataset

[4]note that $or(attr4, attr4)$ cannot be replaced by $attr4$, if $attr4 = 3$ then $or(attr4, attr4) = 1$.

Table VI
BEST GP INDIVIDUALS OBTAINED EXPRESSED IN C PSEUDOCODE

Type	Individual
Best Experiment 1	eq(or(attr4,attr4),neq(attr6,and(attr32,attr35)))
Best Experiment 2	neq(neq(attr4, neq(neq(attr4, attr10), attr10)), neq(attr6, least(and(attr2, and(attr15, not(attr23))), attr35)))
Best Experiment 3	and(grater(neq(or(grater(attr35,neq(grater (or(attr23, attr25), attr36), attr29)), neq(attr14, attr5)),least(attr7, attr35)), eq(attr24, neq(attr14, attr5))), grater(grater(neq(or(attr6, attr37), neq(grater(or(attr23, attr25), attr36), attr29)), least(not(attr12), neq(attr14, attr5))), or(least(grater(attr26, attr13), attr6), attr31)))

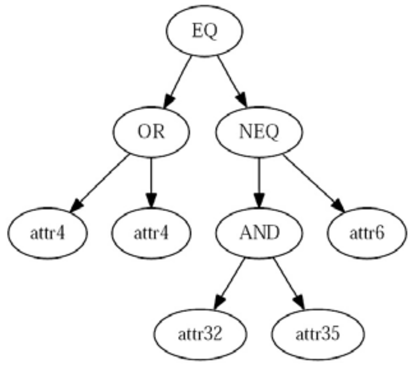

Figure 3. Tree representation of best individual obtained in experiment 1

like *and*, *equal*, etc.). We have not compared execution time of different techniques, as it depends on the implementation and computer where the tests are performed. Instead, we have compared the number of nodes of best GP individuals against state-of-the-art proposals and a C4.5 tree generated by us. Table VII shows that GP generated programs have the fewest number of nodes and function calls.

Table VII
EFFICIENCY COMPARISON BETWEEN STATE-OF-THE-ART PROPOSALS AND BEST GP INDIVIDUALS. GP REDUCES THE SIZE OF THE SOLUTIONS

	Functions	Leaves	Total Nodes
C4.5 Tree	62	332	394
KDD-99 Winner [16]	–	–	500
Faraoun GP [16]	86	–	86
Best Experiment 1	4	5	9
Best Experiment 2	9	9	18
Best Experiment 3	25	25	50

VII. CONCLUSIONS

In this paper we have presented a domain-aware GP approach to the generation of network traffic analysis rules.

We have carefully thought the fitness function and function set in order to produce effective, efficient and easy to understand rules. Thus, two fitness functions, that take into account the IDS evaluation field, have been tested. The well-known KDD-99 dataset has been used as input data for training and testing in such a way that a comparison with state-of-the-art research was possible. Results point out that our approach competes in effectiveness with top proposals and even improve them in realistic scenarios. In addition, generated rules are much shorter. As a consequence, our system is more efficient increasing the throughput while reducing the time to process each incoming event. Finally, these automatically generated individuals can help us to understand why an event is considered intrusive due to their simple structure.

Acknowledgment

This work was partially supported by CDTI, Ministerio de Industria, Turismo y Comercio of Spain in collaboration with Telefónica I+D, Project SEGUR@ with reference CENIT-2007 2004.

References

[1] P. M. Karen Scarfone, *Guide to Intrusion Detection and Prevention Systems*. NIST, February 2007.

[2] J. Quinlan, *C4. 5: programs for machine learning*. Morgan Kaufmann, 2003.

[3] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Škorić, "Measuring intrusion detection capability: an information-theoretic approach," in *Proceedings of the 2006 ACM Symposium on Information*, 2006, pp. 90–101.

[4] A. Orfila, J. M. Estevez-Tapiador, and A. Ribagorda, "Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming," *Applications of Evolutionary Computing*, pp. 93–98, 2009.

[5] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 DARPA/lincoln laboratory evaluation data for network anomaly detection," in *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*. Springer-Verlag, 2003, pp. 220–237.

[6] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, November 2000.

[7] C.-F. Tsai and C.-Y. Lin, "A triangle area based nearest neighbors approach to intrusion detection," *Pattern Recognition*, vol. 43, no. 1, pp. 222 – 229, 2010.

[8] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, December 1992.

[9] M. Crosbie and P. G. Spafford, "Applying genetic programming to intrusion detection," in *Proceedings of the AAAI 1995 Fall Symposium series*. AAAI, 1995, pp. 1–8.

[10] M. Oltean and C. Grosan, "A comparison of several linear genetic programming techniques," *Complex Systems*, vol. 14, no. 4, 2004.

[11] M. F. Brameier and W. Banzhaf, *Linear Genetic Programming (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[12] D. Song, M. Heywood, and A. N. Zincir-Heywood, "A linear genetic programming approach to intrusion detection," in *Genetic and Evolutionary Computation, GECCO 2003*, vol. 2724, 2003, p. 208.

[13] B. Pfahringer, "Winning the KDD99 classification cup: bagged boosting," *SIGKDD Explor. Newsl.*, vol. 1, no. 2, pp. 65–66, 2000.

[14] A. Abraham and C. Grosan, "Evolving intrusion detection systems," in *Genetic Systems Programming: Theory and Experiences*, ser. Studies in Computational Intelligence, N. Nedjah, A. Abraham, and L. de Macedo Mourelle, Eds. Germany: Springer, 2006, vol. 13, pp. 57–80.

[15] A. Abraham, C. Grosan, and C. Martin-vide, "Evolutionary design of intrusion detection programs," *International Journal of Network Security*, vol. 4, p. 2007, 2006.

[16] K. Faraun and A. Boukelif, "Genetic programming approach for multi-categoy pattern classification applied to network intrusions detection," *International Journal of Computational Intelligence*, vol. 6, no. 1, pp. 79–90, 2006.

[17] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data," in *Applications of Data Mining in Computer Security*. Kluwer, 2002.

[18] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Škorić, "Towards an information-theoretic framework for analyzing intrusion detection systems," in *11th European Symposium on Research in Computer Security (ESORICS), Hamburg*. Springer, Sep 2006, pp. 527–546, lNCS Vol. 4189.

[19] C. Elkan, "Results of the KDD'99 classifier learning," *ACM SIGKDD Explorations Newsletter*, vol. 1, no. 2, pp. 63–64, 2000.

[20] I. Graf, R. Lippmann, R. Cunningham, D. Fried, K. Kendall, S. Webster, and M. Zissman, "Results of DARPA 1998 offline intrusion detection evaluation," 1998, DARPA PI Meeting. Cambridge, Massachusetts, USA.

[21] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 227–261, 2000.

[22] H. Kayacik, A. Zincir-Heywood, and M. Heywood, "On the capability of an SOM based intrusion detection system," *Neural Networks, 2003. Proceedings of the International Joint Conference on*, vol. 3, pp. 1808–1813, 2003.