

# A Genetic Algorithm Approach for Doing Misuse Detection in Audit Trail Files

Pedro A. Diaz-Gomez and Dean F. Hougen  
Robotics, Evolution, Adaptation, and Learning Laboratory (REAL Lab)  
School of Computer Science, University of Oklahoma  
Norman, OK, USA  
pdiazg@ou.edu    hougen@ou.edu

## Abstract

This paper focuses on the development of an Intrusion Detection System based on Genetic Algorithms. We present and justify a fitness function independent of variable parameters that addresses the problem of false positives. This fitness function is a generic one that can be used for either off-line or online intrusion detection systems and it allows us consider pseudo intrusions, which could be used to prevent the occurrence of actual intrusions. The paper also describes extending the system to account for the fact that intrusions may be mutually exclusive and defines the union operator which greatly speeds the search for intrusions.

## 1. Introduction

To address the growing problem of data protection, researchers are examining new paradigms in computer security, such as applying evolutionary methods to intrusion detection [6].

### 1.1. Intrusion Detection Systems

An *Intrusion Detection System* (IDS) is a security mechanism that looks for *intrusions*, which are malicious actions focused on gaining access to the resources of the computer [12]. To do that, an IDS must know in advance what the intrusions are—known as *misuse detection*—or it must distinguish between normal and abnormal activity in order to recognize intrusions [12].

Intrusion Detection Systems analyze audit trail records generated by the operating system. The analysis of the audit trail records can be an online process (i.e., done as the records are generated due to actions performed by a user) or an off-line process (i.e., done after the fact) [12]. If done

online, the IDS may be able to recognize malicious actions before the intrusion has been completed.

### 1.2. Genetic Algorithms

*Genetic Algorithms* (GAs) belong to the field of evolutionary computation and have been applied as heuristic methods to solve problems. The strength behind GAs resides in the fact that the search space is traversed in parallel by proposing solutions (at the beginning randomly generated) that are continuously evaluated using a *fitness function*. A GA is a method for moving from one population of solutions to another using a kind of artificial selection and the operators crossover and mutation. Each solution is usually called a *chromosome* or *individual* which consists of *genes* (for binary GAs, bits), each gene being an instance of a particular *allele* (0 or 1). The selection operator chooses those chromosomes in the population that are fittest and that will be allowed to reproduce. Crossover exchanges subparts of two chromosomes. Mutation randomly changes the allele values of some locations in the chromosome [11].

A GA needs a fitness function to be maximized—the combination of objectives and constraints in a single function using arithmetic operators seems to be an appropriate way to define it. This approach is called the *weighted sum approach* [2], that in its general form is

$$\max \sum_{i=1}^k w_i * f_i(\bar{x}) \quad (1)$$

where  $w_i \geq 0$  are the weights representing the importance of  $k$  objectives and constraints which are represented (indistinctly) by  $f_i(\bar{x})$ .

There are, however, problems with this approach. The first is that accurate scalar information must be provided on the range of objectives, to avoid some of them dominating the others. The second is the difficulty in determining appropriate weights when there is not enough information

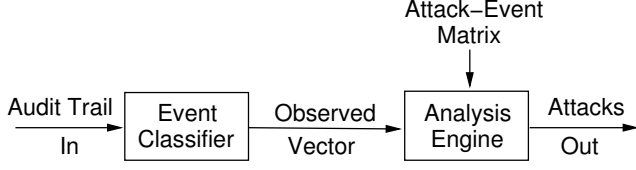


Figure 1. The IDS Architecture

about them. In this case, any optimal point obtained will be a function of the coefficients used to combine the objectives and constraints [2]. We present in this paper a fitness function that follows the weighted sum approach but uses parameters derived from problem itself.

### 1.3. A GA for Misuse Detection

Misuse detection can be accomplished by searching audit trail logs of user activities for matches to patterns of events required for known attacks. Because such search is NP-complete, heuristic methods will need to be employed as databases of events and attacks grow. Genetic Algorithms can provide the basis for an appropriate heuristic search method [10].

Our program to do this builds on ideas found in *GAS-SATA* [10] and consists of two parts (see Figure 1). The first part, the *Event Classifier*, reads the audit trail, classifies audit records into events, and counts the events; the output of this program is an array with a length equal to the number of event types ( $N_e$ ), called the *Observed Vector* ( $OV$ ). The second part, the *Analysis Engine*, is a genetic algorithm that receives as input the Observed Vector and the *Attack-Events matrix* ( $AE$ ), and performs the analysis in order to determine which attacks may have occurred. To do this it repeatedly hypothesizes sets of attacks that could occur and evaluates each hypothesis.

In order to evaluate the hypothesis  $I$ , the attack-event matrix is multiplied by  $I$ . Each position of  $AE * I$  is compared with the corresponding position in  $OV$ . If  $(AE * I)_j$ , (the hypothesized number of events of type  $j$ ) is less than  $OV_j$  (the observed number of events of type  $j$ ) then the hypothesis is possible. But if, on the contrary,  $(AE * I)_j$  is greater than  $OV_j$ , that means that the algorithm hypothesized intrusions that did not occur, a situation that must be taken into account perhaps by using a penalty factor in the fitness function. The penalty term is, then, related to the number of event occurrences in  $AE * I_j$  in excess of the number of event occurrences in  $OV_j$  [10].

## 2. Fitness Function

Following our previous analysis [3], the problem is to find the vector  $I$  that maximizes the inner product

$$F(I) = W \cdot I \quad (2)$$

where  $W$  is the weight vector for the objectives (as in Section 1.2) subject to the constraints [10]

$$(AE * I)_j \leq OV_j \text{ for } 0 \leq j \leq N_e \quad (3)$$

and

$$I_k \geq 0 \text{ for } 0 \leq k \leq N_a \quad (4)$$

where  $N_a$  is the number of known attacks types.

### 2.1. Theoretical Underpinnings

The constraints given in Equation 3 can be arranged as polynomials  $c_j$  in  $i$  following the notation suggested by Ham [8] as

$$c_j(I) = a_{j1}i_1 + a_{j2}i_2 + \dots + a_{jn}i_n - OV_j \text{ for } 0 \leq j \leq N_e \quad (5)$$

where  $a_{jk} \in AE$ . Using the objective function from Equation 2 joined with the constraint from Equation 3 we obtain the *energy function* [8] which is defined as

$$E(I, K) = W \cdot I + K \sum_{j=1}^{N_e} \Phi[c_j(I)] \quad (6)$$

where  $K$  is a positive parameter that controls how the unconstrained optimization problem of Equations 6 to 8 approximates the original *linear problem* in Equations 2 to 4, and  $\Phi[c_j(I)]$  is defined as

$$\Phi[c_j(I)] = \begin{cases} 0 & \text{if } c_j(I) \leq 0 \\ > 0 & \text{if } c_j(I) > 0. \end{cases} \quad (7)$$

$$I_k \in \{0, 1\} \text{ for } 0 \leq k \leq N_a \quad (8)$$

and, as we want to maximize  $E$ ,  $\Phi(t)$  must be differentiable with the property in Equation 7.

For simplicity, the function  $\Phi(t)$  commonly selected is [8]

$$\Phi(t) = \begin{cases} 0 & \text{if } t \leq 0 \\ \frac{1}{2} t^2 & \text{if } t > 0 \end{cases} \quad (9)$$

Finding the partial derivative of  $E$  in Equation 6 with respect to  $I$  we obtain, using Equation 5 and following our previous analysis [3],

$$\begin{aligned} & \sum_{j=1}^{N_e} (a_{j1}i_1 + a_{j2}i_2 + \dots + a_{jn}i_n) * [a_{j1} \ a_{j2} \ \dots \ a_{jn}]^T \\ &= -\frac{W}{K} + \sum_{j=1}^{N_e} OV_j * [a_{j1} \ a_{j2} \ \dots \ a_{jn}]^T \end{aligned} \quad (10)$$



a normalized fitness function equal to

$$0 < \frac{\sum_{j=1}^{N_e} (AE * I)_j - \sum_{j=1}^{N_e} \max[0, (AE * I)_j - OV_j]}{\sum_{j=1}^{N_e} (AE * I)_j} \leq 1 \quad (14)$$

with  $\sum_{j=1}^{N_e} (AE * I)_j \neq 0$ . That corresponds to the definition of the *degree of subsethood of fuzzy subsets*  $S(AE * I, OV)$  [9], leading us to conjecture that every  $AE * I$  vector, each a fuzzy subset of the  $OV$  vector, is an intrusion vector.

Maximums according to Equation 14 are those that have fitness equal to 1, i.e.,  $I$  vectors that satisfy Equation 13,  $\forall j 1 \leq j \leq N_e$ . However, there could be some subset of intrusions that can violate the constraint. Section 3 takes into account this case that corresponds to intrusions that require the same event.

In our previous work [3, 4, 5], we used the fitness function  $F(I) = N_e - T'$  where  $N_e$  corresponds to the total number of classified events and  $T'$  accounts for the penalty and corresponds to the number of overestimates, i.e., the number of times  $(AE * I)_j > OV_j$ . This fitness function and the one defined in Equation 14 have some similarities. In Equation 14 the first term can be simplified and we get 1. The second part, which is the one that takes into account the constraint, is the variable one. However, Equation 14 takes into account the constraint with a greater degree of precision. For example, if in evaluating vector  $I$  we obtain that  $(AE * I)_6 = 11$  and  $OV_6 = 8$ , then  $T' = 1$  because the constraint was violated one time but  $(AE * I)_6 - OV_6 = 3$  in Equation 14. This fact should help to reduce the false positive ratio and this hypothesis will be examined in future research.

### 3. Union Operator

In order to address the problem of more rapidly finding the complete set of intrusions, as our genetic algorithm runs it uses to create an aggregate solution set of all possible compatible types of intrusions found. To do this, it records all the realistic solutions found (those where  $\sum_{j=1}^{N_e} \max[0, (AE * I)_j - OV_j] = 0$ ) in the search space while running and keeps track of each intrusion it finds within each realistic solution. The algorithm then checks if the intrusion already exists in its current solution set and, if it does not, then it checks if it is mutually exclusive or not in order to add it to the corresponding aggregate solution set. In this way, the algorithm builds up the set of all compatible intrusions—the intrusion subset—and the intrusion subset exclusive relative with the previous one. It does this as follows.

An *individual* ( $I$ ) in the population is an array of 0's and 1's. Each position in  $I$  corresponds to a type of intrusion that is being hypothesized by the genetic algorithm. If we consider each intrusion type alone (i.e., one and only one position in  $I$ ), it must satisfy  $\sum_{j=1}^{N_e} \max[0, (AE * I)_j - OV_j] = 0$  for an intrusion of that type to be present. Considering more than one together we have two cases:

1. The constraint is *not* violated, in which case the whole set of intrusions *can* be considered as valid hypothesis. We call this set *the intrusion subset*.
2. The constraint *is* violated (i.e.,  $\exists j |(AE * I)_j > OV_j$ ), in which case the whole set *can not* be considered as a valid hypothesis. So, we check for those that make the hypothesis false, and we call those intrusions *exclusive intrusions*.

The second case occurs for intrusion types that share some event type. For example, intrusions number 5, 19, and 21 share events number 6 and 17 (see Table 1); if the observed vector has, for instance, 8 events of type 6, then intrusion sets  $\{5, 19\}$  or  $\{19, 21\}$  can occur because  $(AE * I)_6 = 6$  but not  $\{5, 19, 21\}$ , because  $(AE * I)_6 = 11$  which violates the constraint  $(AE * I)_6 \leq OV_6 = 8$ . (Other entries must be considered to get full intrusions 5, 19, and 21; we present here only entry 6 for clarity.) So, continuing with this example, if the algorithm first encounters intrusions 5 and 19 then this is the subset of intrusions. Later on, when intrusion 21 appears<sup>1</sup>, it is going to be validated in conjunction with intrusions 5 and 19, but, as stated before, the constraint is violated. 21 is considered, then, an exclusive intrusion. (It could be the case that instead of intrusion 21 appearing, intrusions 21 and 19 appear, in which case the analysis is similar. See Section 2.)

### 4. Empirical Studies

We consider here two empirical studies. The first regards scaling and the second is meant to help determine how effective the union operator is a speeding the search process. For both studies we used 2-tournament selection, 60% crossover probability, and a mutation rate of 2.4%. Normal probabilities are 70% for crossover and 0.1% for mutation [11]; however, as we use only 40 individuals<sup>2</sup>, we increase the mutation rate in order to get more diversity in the population as the algorithm progresses. In all cases, the GAs were allowed to run until all intrusions were found and the metric used, then, was the number of generations until that happened.

<sup>1</sup>Each independent valid hypothesis does not violates the constraint.

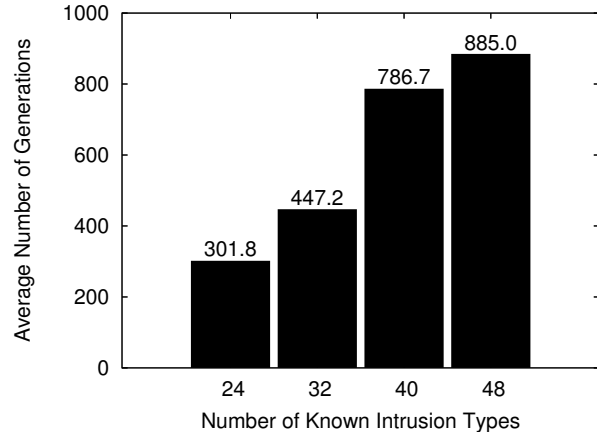
<sup>2</sup>Mé used 100 individuals [10].

Additional Entries for columns 24 – 31									
[0,24]	[1,25]	[2,26]	[3,27]	[4,28]	[5,29]	[6,30]	[7,31]	-	-
100	30	500	40	700	20	30	8	-	-
Additional Entries for columns 32 – 39									
[8,32]	[15,32]	[5,32]	[9,33]	[10,34]	[11,35]	[12,36]	[13,37]	[14,38]	[15,39]
20	4	10	10	500	4	70	20	3	18
Additional Entries for columns 40 – 47									
[16,40]	[17,40]	[18,41]	[19,42]	[20,42]	[21,43]	[22,44]	[23,45]	[24,46]	[25,47]
2	4	10	1	50	4	7	20	3	8

**Table 2. *AE* New Entries as *I* is Enlarged. Experimental Setting.**

<i>OV</i>	<i>I</i> Length			
	24	32	40	48
User2051_7	193.1	343.7	565.2	661.9
User2051_11	220.1	484.3	539.7	814.1
User2506_15	278.6	328.0	925.8	790.9
6 Intrusions	344.3	319.8	843.7	1,129.1
9 Intrusions	291.8	618.8	876.5	759.9
11 Intrusions	356.8	425.5	876.9	900.0
12 Intrusions	428.0	610.3	879.4	1,139.2

**Table 3. Average Number of Generations per User Activity and *I* Length. Union operator used.**



**Figure 2. Average Number of Generations for Different Numbers of Known Intrusion Types.**

#### 4.1. Scaling

As a first attempt to determine if this approach can scale to larger number of known intrusion types, we incrementally expanded the number of known intrusion types from 24 to 48. For doing that, two data structures must be enlarged: the *AE* matrix, which has the attacks, and the intrusion vector *I*, which has the type of intrusion that can occur. So, when we run experiments with 24 intrusion types, we use an *AE* matrix of dimension 28x24 and an *I* vector of length 24 and, when we run experiments with 32 intrusion types, the *AE* matrix is enlarged with the new types of intrusions, and so its size is 28x32 (28 again because the type and number of events (*OV*) is the same in this experimental setup) and the corresponding *I* vector is enlarged to 32. Table 2 shows the new entries used for the *AE* matrix when it is enlarged from 24 – 31, from 32 – 39 and from 40 – 47. The entries were chosen taking into consideration factors like various intrusions sharing the same type of activity, like entries [5, 29] and [5, 32], and a intrusion having more than one type of activity, like intrusion 40 with entries [16, 40] and [17, 40].

70 test runs were carried out for each number of known intrusion types considered; this corresponds to 7 different input data (Observed Vectors *OV*, with fixed length of 28) and 10 repetitions for each one starting with different ran-

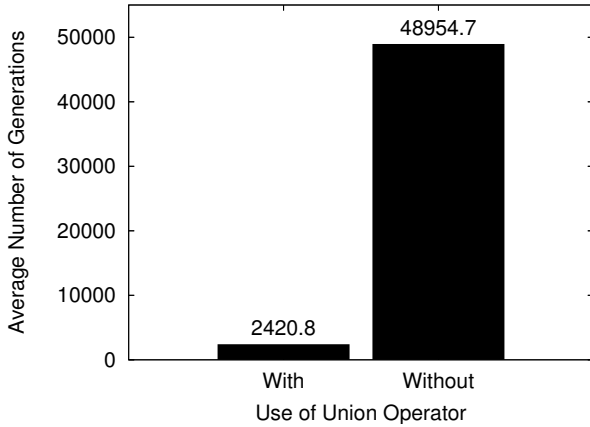
dom initial populations. For example, in Table 3, the first entry, which corresponds to the observed activity of user *User2051\_7* and *I* vector of length 24, is equal to an average of 193.1 over 10 repetitions<sup>3</sup>.

Results are shown in Table 3, where we can see that the maximum number of generations needed was 1,139 for an *I* vector with length 48 and an observed vector containing 12 intrusions. With the data set used the solution was computable. Figure 2 shows the average of the average number of generations taken from Table 3.

#### 4.2. Value of the Union Operator

While heuristic methods are needed to scale up to larger numbers of known intrusion types, not all heuristic methods are equally efficient. In particular, a basic method using a fitness function such as we have proposed which avoids

<sup>3</sup>Data corresponding to users 2051 and 2506 that were extracted by our scanner correspond to audit data files downloaded from the Lincoln Laboratory at MIT [7]; other *OV* vectors in Table 3 are synthetic.



**Figure 3. With Union Operator vs. Without.**

false positives but does not consider false negatives, will not be effective at guiding a normal GA to finding all intrusions in the input data quickly. Quick convergence is the function of the union operator.

To demonstrate this, we repeated the scaling empirical study (Section 4.1 without using the union operator. This gives us a comparison of a fairly typical GA with our own, enhanced GA. The clear conclusion is that the aggregate value of the union operator is that it makes the algorithm converge quickly. If we take the total of each column as in Table 3 and average these together we obtain 2,420.77 generations using the union operator, against 48,954.74 without using it—see Figure 3.

## 5. Subsethood and Pseudo Intrusions

An interesting topic to take into account in doing misuse detection is how near some activity is to being an intrusion, in order to do prevention. The fitness function proposed in Equation 14 allow us to find such *pseudo intrusions*. If we look for individuals such that the constraint is not violated, i.e., the fitness value is equal to 1, then we get a set of intrusions. But if we also look for individuals for which subsethood is

$$\varsigma \leq S(AE * I, OV) < 1$$

with  $\varsigma \gg 0$ , then we obtain all those pseudo intrusions in a neighborhood  $\varsigma$ . The set of pseudo intrusions is exclusive from the set of intrusions, because pseudo intrusions violate the constraint (see Section 3).

As an example, if we take intrusion type 5, which corresponds to column 5 in Table 1, we see that it needs at least

5 occurrences of event 6, 30 occurrences of event 7, and 35 occurrences of event 17.

This means that if the user activity has 5 occurrences of event type 6 and 30 occurrences of event type 7 but only 34 occurrences of event type 17; then *there is no intrusion of type 5* at all. But if, for example, we set the fitness function to have values  $\geq 0.98$ , then we get a *pseudo intrusion of type 5* in this case. This would allow an online IDS to act when the pseudo intrusion is recognized and before the intrusion is successful.

## 6. Discussion of Major Contributions

A genetic algorithm needs a fitness function that combines objectives and constraints into a single value [2]. The problem is not only to find the appropriate form for the function but also to provide accurate values to the parameters so that it will produce the correct solution to as many instances of the problem as possible. Objectives and constraints are in Equation 14.  $\sum_{j=1}^{N_e} (AE * I)_j$  is the *objective*—the more intrusions the better.  $\sum_{j=1}^{N_e} \max[0, (A * I)_j - OV_j]$  is the *penalty* if the *constraint* is violated. However, we highlight that the parameters used were obtained from the problem itself, i.e., there are no parameters to be tuned as suggested in previous research [10].

A *normalized* fitness function has the characteristic that it is good for finding local minima/maxima. Knowing this fact, we propose the use of the *union operator*, so that each time the algorithm finds a local maxima it is stored—avoiding false negatives—and it is tested in conjunction with previous maxima obtained. If in the testing there is no violation of the constraint, then the new local maxima belongs to the intrusion subset, but if the constraint is violated, the new local maxima belongs to the set of exclusive intrusions, as stated in Section 3. We have performed an empirical study in order to test the validity of this assumption, and found a speed up well in excess of an order of magnitude [5].

A normalized fitness function can also be used to measure *pseudo intrusions*, so if the fitness function in Equation 14 is used in an online system, then the prevention of intrusions can be accomplished.

The work developed in this article extends and validates our previous work [3, 4, 5] using intrusion vectors of length 32, 40, and 48 as shown in Section 4.1.

## 7. Conclusions and Future Work

This paper expands previous work and proposes a fitness function for doing misuse detection and finding pseudo intrusions. We conjectured that every validation of the vector  $I$  with the Attack-Event matrix  $(AE * I)$  fuzzy subset of the

observed vector  $OV$  is an intrusion vector. We introduce the concept of the union operator and show experimentally how it improves the convergence of the algorithm in terms of number of generations. We introduce the concept of the intrusion set and exclusive intrusions so the algorithm dis-aggregates possible subsets of misuse of the system.

More theoretical and experimental work can be done in order to assure that there are no intrusions. The concept of pseudo intrusion can be refined because currently, if an intrusion has small values in the  $AE$  matrix, then that intrusion will be in the neighborhood  $\zeta$  for almost all the observed vectors  $OV$ . This is an open problem for us as is the number of generations needed in order to get the complete set of intrusions.

We expanded our previous work up to 48 intrusions types. However, the system should be enlarged to hundreds or thousands intrusion types, in order to validate its performance in realistic scenarios. Besides this, the system should be compared with similar ones or with other approaches.

Many approaches have been explored for doing intrusion detection. However, commercial products confine themselves to traditional methods like statistical characterization of user activity and usage patterns [1]. This paper presents some of new research in intrusion detection, by using a modified GA as an analytical engine that performs intrusion detection.

## References

- [1] R. G. Bace. *Intrusion Detection*. MacMillan Technical Publishing, USA, 2000.
- [2] C. A. C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1998.
- [3] P. A. Diaz-Gomez and D. F. Hougen. Analysis and mathematical justification of a fitness function used in an intrusion detection system. In *Proceedings of the Seventh Annual Genetic and Evolutionary Computation Conference*, pages 1591–1592, 2005.
- [4] P. A. Diaz-Gomez and D. F. Hougen. Analysis of an off-line intrusion detection system: A case study in multi-objective genetic algorithms. In *Proceedings of the Florida Artificial Intelligence Research Society Conference*, pages 822–823, 2005.
- [5] P. A. Diaz-Gomez and D. F. Hougen. Improved off-line intrusion detection using a genetic algorithm. In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 66–73, 2005.
- [6] P. A. Diaz-Gomez and D. F. Hougen. Three Approaches to Intrusion Detection: Analysis and Enhancements. In *Proceedings of the sixth National Computer and Information Security Conference ACIS-Colombia*, 2006.
- [7] D. Fried and M. Zissman. Intrusion detection evaluation. Technical report, Lincoln Laboratory, MIT, 1998. <http://www.ll.mit.edu/IST/ideval/>, accessed March 2004.
- [8] F. M. Ham and I. Kostanic. *Principles of Neurocomputing for Science & Engineering*. Mc Graw Hill, 2001.
- [9] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice Hall, USA, 1995.
- [10] L. Mé. GASSATA, a genetic algorithm as an alternative tool for security audit trail analysis. In *First International Workshop on the Recent Advances in Intrusion Detection*, Belgium, 1998.
- [11] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.
- [12] B. C. Tjaden. *Fundamentals of Secure Computer Systems*. Franklin and Beedle & Associates, 2004.