GA-NIDS: A Genetic Algorithm based Network Intrusion Detection System

Anup Goyal
Electrical Engineering and Computer Science
Ford 2-207, Northwestern University
Evanston, Illinois
anup-goyal@northwestern.edu

Chetan Kumar
Electrical Engineering and Computer Science
Tech L-460, Northwestern University
Evanston, Illinois
chetan-k@northwestern.edu

ABSTRACT

Detection of malicious connections in computer networks has been a growing problem motivating widespread research in computer science to develop better intrusion detecting systems (IDS). In this paper, we present a machine learning approach known as Genetic Algorithm (GA), to identify such harmful/attack type of connections. The algorithm takes into consideration different features in network connections such as type of protocol, network service on the destination and status of the connection to generate a classification rule set. Each rule in rule set identifies a particular attack type. For this experiment, we implemented a GA and trained it on the KDD Cup 99 data set to generate a rule set that can be applied to the IDS to identify and classify different types of attack connections. Through our experiment, we have developed a rule set comprising of six rules to classify six different attack type of connections that fall into two classesn namely denial of service and probing attacks. The rule generated works with 100% accuracy for detecting the denial of service type of attack connections, and with appreciable accuracy for detecting the probe connections. Results from our experiment have given promising results towards applying genetic algorithm for network intrusion detection.

1. INTRODUCTION

Intrusion detection to identify attacks on computer systems has been a challenging problem in the domain of network security for quite some time. Software to detect network intrusions protects a computer network from unauthorized uses thereby preventing malicious activities. The intrusion detector-learning task is to build a classifier (i.e. a predictive model) capable of distinguishing between attack/intrusion ("bad" connections), and normal or good connections.

Considering the growing problems in network security and the need to develop sophisticated and robust solutions, the KDD Cup was organized in 1999 inviting researchers across the world to design innovative methods to construct an IDS on a training and testing data set, popularly referred to as the KDD Cup 99 data set [3]. Since then, different machine-learning techniques such as Bayesian Classifiers and Decision Trees have been trained on the KDD Cup 99 data set to learn normal and inconsistent patterns from the testing data and thus generate classifiers that are able to detect an intrusion attack [1]. The problem of network intrusion detection is not just to identify the attacks connections, but also to know what type of an attack the connection belongs to [3]. The limitation of these classifiers is that they generate a unified rule for all the attack types. As a result, although the algorithms perform well in segregating attacks from the normal connections, their performance is not so appreciative when it comes to identifying what type of an attack the connection is/was. This can be intuitively explained by the fact that a single rule cannot accurately classify all the attack types. This essentially forms the problem statement and the motivation behind this project. We would like to have an intrusion detection system that has the power to also predict the type of incoming attacks other than identifying attack connections. Amongst the different machine learning techniques, we selected Genetic Algorithm as a good way to find an efficient solution to the problem.

Genetic Algorithms are another machine learning approach based on the principles of evolutionary computation [4]. They incorporate the concept of Darwin's theory and natural selection to generate a set of rules that can be applied on a testing set to classify intrusions. Researchers have explored the use of GAs in intrusion detection, and reported very high success rates, but on data sets other than the KDD 99 Cup, such as the DARPA data set [2][4][5]. We have designed a GA that generates a set of rules, wherein each rule has been evolutionary generated to classify every attack type in the training and testing data set. The rationale behind selecting GA for this task is due to the inherent evolutionary trait in the algorithm which allows us to define our own fitness function based on which only those members or rules or hypothesis are selected that satisfy our fitness criterion (described later). We can therefore make the algorithm select only those individuals that best classify attack types at every step of its evolution.

With this approach in mind, we have designed and implemented a GA with the focus to achieve a high prediction accuracy rate >95% and minimal false positive rate on the KDD Cup 99 data set. This prediction accuracy rate sets the benchmark for us to test the performance of the GA. In this paper, we present details on the design of the GA, experimental set up and results from our implementation. So far, out of the 24 different attack types in the training data set, we have been able to generate a rule set containing six different rules belonging to two separate classes (explained in next section), wherein each can correctly classify the attack labels that it has been selected to correctly identify, and thus subsequently predict. The training of the algorithm was carried out on the 10% KDD Cup training data set (explained in next section), while the testing has been carried out on the entire 100% labeled set. The combination of rules gives us an appreciable accuracy (upto 98%) on the training data set and 92% accuracy on the testing data set thus justifying the choice, performance and applicability of Genetic algorithms to network intrusion detection.

2. TRAINING AND TESTING DATA SET

We have used the KDDCUP 99 data set to train and test the system classifier. The dataset has been provided by MIT Lincoln Labs. It contains a wide variety of intrusions simulated in a military network environment set up to acquire nine weeks of raw TCP/IP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. The LAN was operated as if it were a true Air Force environment, peppered with multiple attacks. Hence, this is a high confidence and high quality data set.

They set up an environment to collect TCP/IP dump raws from a host located on a simulated military network. Each TCP/IP connection is described by 41 discrete and continuous features (e.g. duration, protocol type, flag, etc.) and labeled as either normal, or as an attack, with exactly one specific attack type (e.g. Smurf, Perl, etc.). Attacks fall into four main categories: (i) Denial of Service Attacks (DOS) in which an attacker overwhelms the victim host with a huge number of requests. (ii) User to Root Attacks (U2R) in which an attacker or a hacker tries to get the access rights from a normal host in order, for instance, to gain the root access to the system. (iii) Remote to User Attacks (R2L) in which the intruder tries to exploit the system vulnerabilities in order to control the remote machine through the network as a local user. (iv) Probing in which an attacker attempts to gather useful information about machines and services available on the network in order to look for exploits. For our system, we have used 10% of the training set containing as published by Lincoln Labs which contains 494,021 connections. Our testing set is the entire set of labeled connections consisting of around 4.9 million connections. Thus, using the entire data set we have been able to test our system on unseen connections.

Till the current stage of implementation, we have been able to generate a rule set comprising of six rules, each to correctly classify six different attack labels. We selected the top three distributions of attack labels from the two classes of attacks: DOS and Probe, in the 10% training data set. These six attack labels are: smurf, Neptune, land: type of DOS attacks and satan, ipsweep and portsweep: type of Probe attacks.

3. NID ARCHITECTURE

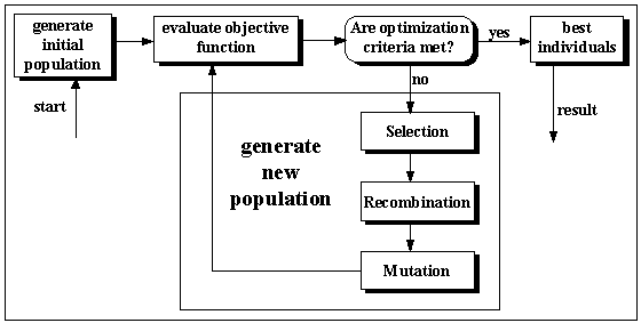

Figure 1: The NID architecture for our first stage implementation.

Figure 1 represents the higher-level architecture of the IDS. Based on our fitness criteria, we select the best-fit individuals from every population that are able to detect the attacks and classify them correctly into their respective sub-attack labels i.e. smurf etc. We then apply crossover and mutation on them to generate new rules that are more biased towards detecting the attacks. Over a series of generations, we generate a rule base which comprises of one rule to correctly classify the type of attack that it has been trained to classify. When this project is completed, given that there are 24 attack labels in the training data set, our rule base shall contain 24 different rules to classify each attack type. Till this stage of implementation, as mentioned earlier, our rule base correctly classifies six different types of attacks.

4. METHODS

4.1 ENCODING OF CONNECTIONS

As mentioned before, every network connection in the KDD Cup data set contains 41 fields/features out of which, four including the label are strings, 14 fields carry float type values in the range 0.00-1.00, while the remaining 23 fields carry integer values. With 41 different fields having varying types of values, the hypothesis search space i.e. number of possible individuals in every population becomes unfathomably large and will thus necessitate a very large population size requiring ultra-scale computing to select fit individuals. Due to this, we restricted our hypothesis space by calculating the upper and lower bounds for every field from the training set and allowed for values only within this range in the random population we generate at the start of the experiment. We did so by using linux console commands to find the maximum and minimum values for each field. The fields that carry string type of values were represented using positive numbers starting from 0, with every new string type assigned an incremented number. For example, for the 'protocol' field, icmp = 0, udp = 1. Other fields carrying float type of values were incorporated in the genotype i.e. the encoded string, as they appear in the connection. As per GA principles, at the start of the experiment, we create an initial population that has individuals whose genes are selected at random from the range that they van vary in. Every population comprises of 100 individuals.

4.2 FITNESS FUNCTION

After generating the initial population, we will use the fitness function as a metric to select the fit individuals who would undergo crossover and mutation to create the next generation population. Our fitness function is given by the formula: **$F = a/A - b/B$,** where a represents the number of attack connections the individual correctly classifies out of a total of A attacks, b represents the number of normal connections a network correctly classifies out of a total of B normal connections in the population. Hence, we can see that our fitness function value would lie in the region [-1,1]. A positive value will denote that the individual classifies more number of attacks correctly than it does the normal ones. To select the fit individuals, we have set a threshold value of 0.95. Thus, all individuals that have a fitness score > 0.95 are selected to produce subsequent generations and are deemed fit. The design of the fitness function is such to make it biased towards individuals that correctly classify only the attack connections, since this is the objective. Ideally, we select the 60 fittest individuals to undergo crossover and produce the next generation. But if a situation arises wherein the number of fit individuals are less than 60, we reproduce the as many fit individuals to create a parent population of 60. To complete the set of 100 individuals in a population, the top 40 offsprings from the 60 are duplicated.

4.3 CROSSOVER AND MUTATION

The fit individuals selected based on their fitness score are now made to undergo crossover to generate new rules or hypothesis to correctly classify attack connections that might not be there in the training data set. We have implemented only single point crossover. An example of how this operation works is as follows: The characters in bold will get crossed over. '#' is a separator to denote different fields.

String1: 000#0256#**1169**; String2: 010#1000#**1459**

Child1: 101#256#**1459**; Child2: 000#000#**1169**.

Crossover happens only at the interface of separation of two fields, and not between them. This has been done to ensure that the resulting offsprings have values that maintain the range of allowable values for every field. In order to search the entire hypothesis space, we also mutate the offsprings to generate new rules or hypothesis. We have used a single field mutation strategy wherein any one field in the offspring is mutated. The mutation rate has been set 1% i.e. out of 100 individuals in a population; only 1 individual will undergo mutation.

5. EXPERIMENTAL SET UP

4.1 OBJECTIVE

As mentioned before, the scope of our experiment was focused to generate classifiers or rules for six attack types belonging to two different classes. The training set contains maximum connections of the smurf attack type, 280,790 to be precise. The number of other connections are: 107201 neptune connections, 21 land connections, 15 satan connections, 30 ipsweep connections and 15 portsweep connections. Hence, we wanted to create a rule that can classify all of these connections with a minimal false positive rate. Although we would have preferred to extend our implementation to all the attack types and connection features, the enormous training time complexity of the algorithm, very large data sets and lack of time restricted us.

4.2 TOOLS

For our implementation, we have used the GALIB C++ library especially suited to develop GAs [12]. Owing to the large hypothesis search space and high time complexity, we wanted to use a tool or library that is high on performance and computing speed. After a comprehensive survey of many tools, we decided to use GALIB since it is a C++ library, has been widely used by other researchers and well documented. We used a LINUX based Dell computer with a Pentium 4 processor, 120GB of hard disk space and 1 GB of RAM to execute the computer program.

4.3 HYPOTHESIS SEARCH SPACE

In this experiment, we restricted our hypothesis search space to the eight most important fields that we could identify. Although, we can extend our search space to all 41 fields, that will however require the computation to go over many hours. Hence, for our experiment we confined our search space to only eight important fields. We selected these fields based on a heuristic analysis of the training data to identify potential fields that seemed unique to a particular attack type. These fields provide information on the type of protocol, type of service, flag (error or normal connection), duration of bytes sent, duration of the connection, % of connections to different hosts, number of operations on access control files and number of outbound commands in an ftp control session features of the connection. From our analysis of the training data, these fields appear to be amongst the decisive fields that can help identify an attack from a normal connection. We intend to extend our implementation to more fields for next experiments. The population size for each generation contains 100 individuals.

4.4 GENETIC OPERATORS and PARAMETERS

We say that an individual matches an attack type when all the eight fields that constitute our search space of the individual match those of the attack connection. The rate of crossover was set to 0.6 i.e. given 100 individuals in any population, 60 best fit individuals will be selected based on highest fitness score and be made to undergo crossover to create offsprings. Since at present we are exploring only eight fields, the crossover occurs only over these fields. Out of the 60, the best 40 parents are then selected to complete the population size of 100. Thus, the best fit parents also participate in the subsequent generations. The mutation rate has been affixed to 1% wherein, only 1 individual out of a population size of 100 undergoes a change in one of the eight fields as explained in section 4.3.

5. RESULTS

From the above experiment, we were able to create a rule that could successfully classify all of the 280,790 smurf type of attack connections. Along with this, it also classified 410 normal connections as attack. The false positive rate is thus around 0.08%**.** In the entire testing data set, the smurf rule set correctly classified 2,807,880 connections, and with a false positive of 0.17%. The rule set that classifies Probe attacks was able to correctly classify 52 Probe attack connections in the training data set, out of a total of 60 probe type of attacks. The rule set for Probes on the entire test data showed results as follows: total number of probe attacks = 38,786, total classified correctly = 35,829. The percentage accuracy = 92.3%. These are encouraging results considering we have used only 8 fields. All these results are shown in Figure 2.

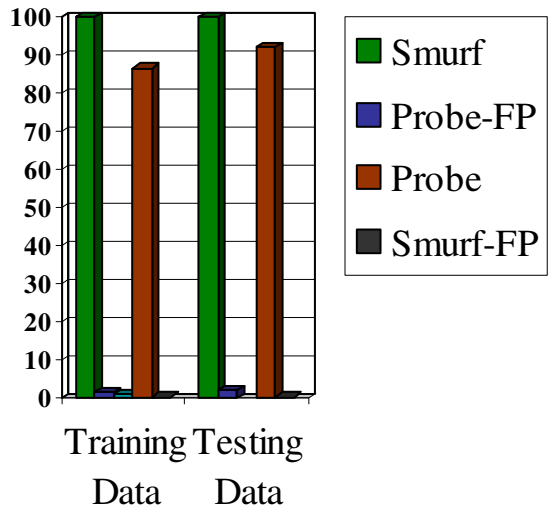

Figure 2: Graph showing percentage accuracy of the rule base for the two attack classes namely smurf (DOS) and probe attacks, along with the false positive rates denoted by FP.

6. CONCLUSIONS

Our results have been very encouraging. We were able to generate a rule using the principles of evolution in a GA to classify all types of smurf attack labels in the training data set. Our false positive rate is also quite low at 0.2% and accuracy rate is as high as 100%. These results have provided us with the impetus to extend our program and apply it to search over all the fields in the connections. We hope this would improve the performance of the GA considering that at present we are able to classify all of the smurf attack type of connections based on matching only eight fields.

We had planned to apply Boosting to our system to enhance the performance. Although, boosting can be applied effectively to minimize the error in a IDS, at present we have not incorporated it. The overall complexity of this project was quite a challenge to cope up with to allow room for us to implement Boosting.

7. FUTURE WORK

Other than searching over the entire hypothesis space of 41 features, there are certain areas, which if looked into can improve the performance of our system. At present, our fitness criterion to select fit individuals is based on the eight fields that match exactly with the attack connection. One can create a statistics driven metric to find the range of values for each of these eight fields that would best classify the attack labels. One could also use data mining techniques to identify the most significant fields that can identify the attack connections. This would add more scientific reason to the search space of the algorithm.

ACKNOWLEDGEMENTS

We would like to extend our thanks and gratefulness to Prof. Bryan Pardo (Professor, EECS, Northwestern University) for our fun filled enriching experience in the Machine Learning course instructed by him. We also thank our class friends namely Jinyu, Alex, Nick and others who reviewed our paper and provided us with valuable suggestions.

REFERENCES

[1] Amor et al. Naïve Bayes vs Decision Trees in intrusion detection systems, *ACM Symposium on Applied Computing*, 2004.

[2] Wei Li. Using Genetic Algorithm for Network Intrusion Detection. *Proceedings of the United States Department of*

[3] KDDCUP 1999 http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[4] Vladimir, M., Alexei, V., and Ivan, S. The MP13 approach to the KDD'99 classifier learning contest. *SIGKDD Explorations, 2000 ACM SIGKDD. 1*(2). January 2000.

[5] Crosbie, M., and Spafford, G. Applying genetic programming to intrusion detection. In *Proc.1995 AAAI Symposium on Genetic Programming*, pp. 1-8.

[6] Pfahringer, B. Winning the KDD99 Classification Bagged boosting. *SIGKDD Explorations, 2000 SIGKDD. 1*(2), pp. 65-66, January 2000.

[7] Dasgupta, D., and Gonzalez, F. A. An intelligent decision support system for intrusion detection and response. In *Proc. Int'l Workshop on Mathematical Methods, Models and Arch.For Computer Networks Security*, pp. 1-14, 2001.

[8] Ghosh, A., and Schwartzbard, A. A study in using neural networks for anomaly and misuse Detection. *8th USENIX Security Symposium*, pp. 141-151, 1999.

[9] Saniee M., Habibi J., Lucas C. Intrusion detection using a fuzzy genetics-based learning algorithm. *Journal of Network and Computer Applications*, 30(1), pp. 414 – 428. January 2007

[10] *Energy Cyber Security Group 2004 Training Conference*, Kansas City, Kansas, May 24-27, 2004.

[11] Adithya Chittur. Model generation for an intrusion detection system using genetic algorithms, Ossing, NY. 2001

[12] GALIB, a c++ library of GA components, http://lancet.mit.edu/galib-2.4/