Using Grammatical Evolution for Evolving Intrusion Detection Rules

DOMINIC WILSON, DEVINDAR KAUR
Electrical Engineering and Computer Science
University of Toledo
Toledo, Ohio
USA

Abstract: - The use of Grammatical Evolution for automating the intrusion detection rules is investigated in this paper. We apply this method to the KDD99 intrusion dataset and demonstrate its usefulness in this context. We achieve favorable results by evolving rules for classifying both normal and abnormal traffic.

Key-Words: - Grammatical Evolution, Intrusion Detection, Rule-based inference.

1 Introduction

An intrusion is defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource. Intrusion detection is the process of monitoring and analyzing events for signs of intrusion [1]. Intrusion Detection Systems (IDS) have become a standard component of network security infrastructures as they allow network administrators to detect policy violations. These violations can range from unauthorized users trying to gain access to misuse of resources in order to deny legitimate access. Detecting such violations is a necessary step in taking remedial action.

In order for an IDS to detect intrusions it needs some method of differentiating normal network use from abnormal use, in this paper Grammatical Evolution is used to derive classification rules using the set of 41 attributes of the KDD99 dataset.

Grammatical Evolution (GE) is a method of evolving syntactically correct programs in an arbitrary context free language. GE has been tested with successful results on a symbolic regression problem, a trigonometric identity problem and a symbolic integration problem.

GE makes use of Genetic Algorithms as its method of finding solutions. In Genetic Algorithms use is made of selection, mutation and recombination operators to evolve the fittest genome for creating the program. Genetic Algorithms are introduced in the next section, prior to a discussion of the Grammatical Evolution, and the GE techniques used in this investigation. Section 4 explains the intrusion dataset used. Our approach and results are stated in section 5. We end with our conclusion and future direction.

2 Genetic Algorithms

Genetic Algorithms are forms of search procedures that use analogies of the genetic operations found in biology [2]. Darwin's theory of natural selection explains the survival of the fittest organisms in a population. The result is that genes encoding traits that favor fitness will be passed down through the generations, until they become common. Furthermore, when two organisms mate and produce offspring, characteristics of each parent are combined in new ways; so it is possible that the offspring will inherit features from both parents and be better suited to the environment Also rare genetic mutation will result in some individuals possessing traits that were not present in the population before.

In genetics, the basic structures are lengths of DNA called chromosomes. These are made up of a number of genes, each of which encodes a particular trait such as eye color; the range of 'values' that the gene can take are called the alleles. The particular combination of alleles on a chromosome can be thought of as a blueprint for an individual and is referred to as a genotype. The resulting physical characteristics of an organism upon expression of the genotype are known as the phenotype. The structures manipulated by a Genetic Algorithm are simplified model of these chromosomes that are usually made up of binary strings. The aim of Genetic Algorithm is to develop an optimal genotype by applying the simple genetic operators of selection, crossover and mutation. Genetic Algorithm fitness is usually measured as a function of how each phenotype measures up to a particular standard.

Selection is the process of determining which individuals will become parents for the next generation. The probability of an individual going on to breed is made some way proportional to its

current fitness. This results in a 'mating pool' biased towards individuals of higher fitness. In order to simulate the genetic recombination that occurs naturally during sexual reproduction, some individuals will be used for 'breeding' via the crossover operator. A predefined crossover probability determines how often this operator is applied. In the single-point crossover used in this investigation, each parent string is divided into a 'head' and 'tail' section, and the tails are swapped over. For the mutation operation, each position in the binary string has a very small probability (typically 0.001) of having its value randomly altered.

3 Grammatical Evolution

Grammatical Evolution (GE) [3,4,5] is a method of evolving syntactically correct programs in an arbitrary context free language. The language to be generated is specified using a Backus Naur Form (BNF) grammar, consisting of a series of production rules mapping a set of non-terminal symbols to the set of terminals that are defined in the language.
Backus Naur Form (BNF) is a convention for expressing the grammar of a language in the form of production rules. The BNF grammar can be represented by a tuple, {N, T, P, S}, where N is the set of non-terminals, T is the set of terminals, P is the set of production rules that map the elements of N to T and S is the start symbol which is an element of N.

An example BNF grammar is displayed in. In the example BNF grammar, the symbol "::=" denotes that the non-terminal on the left of the production rule can be mapped into the symbol that appears on the right. The pipe symbol | is used to denote 'or'. Definitions can be recursive as the first production rule in shows.

3.1 How Grammatical Evolution works

A linear genome made up of a variable number of 'genes' – each of which is represented by an 8-bit binary number – is used to control how the BNF grammar definition is mapped to an actual program. A Genetic Algorithm involving the use of selection, mutation and recombination is used to evolve the fittest genome for creating the program. The evolutionary aspect of GE is language independent, and can theoretically be used to generate functions of arbitrary complexity.

```
N = {statement, condition, function, variable, value}

T = {x,y,>,a,b,c,d,f1,f2,f3,f4,f5 }

S = <statement>

P =

<statement>     ::=IF <condition>  <statement> Else
        <statement> END;        (0)
                | <function>    (1)
<condition>     ::= (<variable> > <value>)
<variable>      ::=x            (0)
                | y             (1)
<value>         ::= a           (0)
                | b             (1)
                | c             (2)
                | d             (3)
<function>      ::=f1           (0)
                | f2            (1)
                | f3            (2)
                | f4            (3)
                | f5            (4)
```

Fig. 1: Example BNF production rules

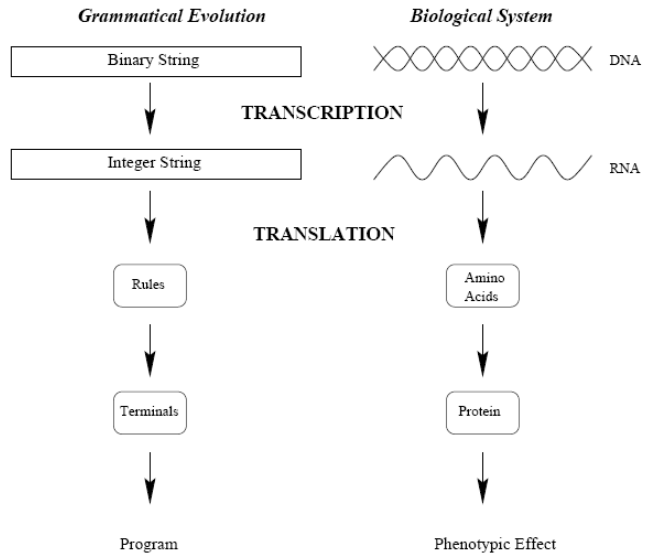

Fig. 2: Grammatical Evolution System and Biological System genotype to phenotype mapping [6]

Fig. 2 outlines the mapping process used in both GE and biological systems. In this implementation, the binary string is segmented into 8 bit sections called codons. The codons are transformed to their corresponding integer values and these values used to decide which expansion to carry out whenever there is a choice between two or more possible expansions. A rule is selected by using the following [6]:

(Integer Codon Value) Mod (Number of production rules for current non-terminal)

Consider for example the following rule:

<value>::= a (0)
| b (1)
| c (2)
| d (3)

i.e. given the non-terminal "value" there are four production rules to select from. If we assume the integer corresponding to the codon being read is 29, then 29 MOD 4 = 1 resulting in the selection of "b" as the terminal. The system traverses the genome by reading a new codon each time a production rule has to be used.

104, 68, 44, 216, 17, 61, 123, 230,217, 71,250, 19, 62, 159, 122, 201, 123
Fig. 3: Example codon sequence

Consider a chromosome with the codon sequence expressed in Fig. 3 (expressed in decimal). The mapping process starts from the start symbol <statement>; because there is more than one production for this symbol (i.e. 2), we use the stated formula to get 104 MOD 2 = 0, and choose the production labeled (0). <statement> is replaced with:

IF <condition> <statement> Else <statement> END;

The next step involves the replacement of the leftmost non-terminal "<condition>" by "(<variable> > <value>)". Because there was only one production for "<condition>", a new codon value was not used. The current expression becomes:

```
IF (< variable> > <value>)
        <statement>
Else
        <statement>
END;
```

Calculating 68 MOD 2 = 0, specifies the use of the production:

<variable> ::=x.

Full resolution of the codon sequence yields the program:

```
IF(x>a)
        IF(y>b)
                (f1),
        ELSE
                (f2),
        END;,
ELSE
        IF(y>c)
                (f3),
        ELSE
                (f4),
        END;,
END;
```

The mapping system can be seen as having transformed a binary string into a program that partitions a feature space into a decision tree. The fitness of the program (and its equivalent decision tree structure) is assessed by substituting feature values and measuring the difference between the calculated output and observed output of the process being identified.

4 The KDD 99 intrusion datasets

In 1999, recorded network traffic from the DARPA 98 Lincoln Lab dataset[7] was summarized into the KDD 99 dataset with 41 atributes per record. We ran our experiments using the 10% KDD 99 intrusion dataset. These have been part of the de facto standard for training and testing intrusion detectors since they were published [8].

Attack Class	**Specific attack**
DoS	back, land, neptune, pod, smurf, teardrop, apache2, mailbomb, processtable, snmpgetattack, udpstorm,
Probe	ipsweep, nmap, portsweep, satan, mscan, saint,
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster, httptunnel, named, snmpguess, worm, xlock, xsnoop, sendmail,
U2R	buffer_overflow, loadmodule, perl, rootkit, sqlattack, xterm, ps,

Table 1: Classification of attacks

The training dataset contained about 5,000,000 connection records, and the training 10% dataset consisted of 494,021 records among which there were 97,278 normal connections (i.e. 19.69%). The data set has 41 attributes for each connection record

plus one class label as shown in Table 1. The 39 different attack types present in the 10% datasets are given in table 1. The attack types can be classified into four main categories as shown in Table 1:

Denial of Service (DoS) is a class of attack where an attacker makes a computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate users access to a machine.

Variable name	Variable type
Duration	continuous
protocol type	discrete
service	discrete
flag	discrete
src bytes	continuous
dst bytes	continuous
land	discrete
wrong fragment	continuous
urgent	continuous
hot	continuous
num failed logins	continuous
logged in	discrete
num compromised	continuous
root shell	continuous
su attempted	continuous
num root	continuous
num file creations	continuous
num shells	continuous
num access files	continuous
num outbound cmds	continuous
is host login	discrete
is guest login	discrete
count	continuous
srv count	continuous
serror rate	continuous
srv serror rate	continuous
rerror rate	continuous
srv rerror rate	continuous
same srv rate	continuous
diff srv rate	continuous
srv diff host rate	continuous
dst host count	continuous
dst host srv count	continuous
dst host same srv rate	continuous
dst host diff srv rate	continuous
dst host same src port rate	continuous
dst host srv diff host rate	continuous
dst host serror rate	continuous
dst host srv serror rate	continuous
dst host rerror rate	continuous
dst host srv rerror rate	continuous

Table 2: Variables for intrusion detection data set

A remote to user (R2L) attack is a class of attack where an attacker sends packets to a machine over a network, then exploits the machine's vulnerability to illegally gain local access as a user.

User to root (U2R) exploits are a class of attacks where an attacker starts out with access to a normal user account on the system and is able to exploit vulnerability to gain root access to the system.

Probing is a class of attack where an attacker scans a network to gather information on known vulnerabilities. An attacker with a map of machines and services that are available on a network can use the information to look for exploits.

5 The use of GE on KDD 99 dataset

The GE based classifier design involves:

1) Designing a grammar (in BNF form) that dictates the syntax of the partitioning and rule structure.

2) Determining a fitness function for using GE to optimize the classifier, and

3) Evolving the candidate structures until some predetermined stopping criteria is reached.

Non_terminals = {<statement>,<termlist> , <c_operator>, <d_operator>, c_value,d_value }

Terminals = {variables = (Duration , Src_byte,…, dst host srv rerror rate), <>,=.>,<}

S = <statement>

P=

<statement> ::= SELECT Count(*) FROM kdd_data WHERE (<termlist> AND <termlist>) AND (Class = DOS)

<termlist> ::= (<termlist> AND <termlist>)
|<c_variable> <c_operator> <c_value>
|<d_variable> <d_operator> <d_value>
| Duration <c_operator> <c_value>
|Src_byte <c_operator> <c_value>
| protocol type <d_operator> <d_value>
| service <d_operator> <d_value>
.
.
.
| dst host srv rerror rate <c_operator> <c_value>

<d_operator> ::= ‘<>’
| ‘=’

<c_operator> ::= ‘<’
| ‘>’

Fig. 4: BNF used

The design objective is to efficiently search for compact classifiers that are highly interpretability have high classification accuracy.

The BNF of the GE grammar was designed to produce relational statements that were used to obtain data from the 10% KDD dataset which was stored in a relational database. The general form of the BNF is shown Fig 4.

c_operator and d_operator refer to continious and discrete operators respectively. For continious variables, the continuous operators '>' and '< are used, and for discrete variables the operators '=' and '<>' were used. C_value's are values that equally divide the domain of each artribute into four and d_values correspond to the actual discrete values relevant to each discrete variable.

An example SQL Select statement that can be produced by this BNF is:

```
SELECT Count(*) FROM kdd_data
WHERE (((protocol_type <> 'udp')
AND (serror_rate < 0.75))
AND ((Class = Normal));
```

In this example protocol type is a discrete attribute and udp is a specific protocol type; serror_rate is a continuous variable and 0.75 is ¾ of the range of serror_rate which spans 0 to 1.

Assessing the fitness of each SQL Select statement was done by:

$$fitness = \frac{A \cap B}{A \cup B} \qquad (1)$$

Where A is the join of the attributes and B is a specific output class. Using our example SQL statement,

$$A = \{x \mid x \in (prototype <> udp) \wedge (serror\_rate < 0.75)\}$$

$$B = \{x \mid x \in (class = normal)\}.$$

Selections were made based on the rank of the fitness. The population size was set at 300 individuals; mutation was set at 1% and the single point crossover rate at 80%. The classification accuracy obtained for different attack types are shown in Table 3.

6 Conclusion

The potential for developing intrusion system rules using Grammatical Evolution is demonstrated. The GE method is a form of symbolic regression and can work with both continuous and discrete input attributes without any assumption that the discrete values form a continuum. The accuracy is higher among the Denial of Service and Normal classes which form a large proportion of the training dataset (79.2% and 19.7% respectively). As with many other results reported eg [9] the attacks involving content (ie U2R and R2L), show low detection rates. The current work is limited to finding the single hypercube of attribute properties which best infers a class. We are looking at extending this work into a union of hypercubes such that non-linear separation into classes can be better achieved. Further work will also include the development of hierarchical rulebased intrusion detection systems using GE.

Class	Training Data Accuracy (%)	Testing Data Accuracy (%)
Normal	87.6	54.6
Dos	96.2	96
Probe	59.1	49.4
U2R	45.6	31
R2L	23.6	1.4

Table 3: Classification accuracy obtained

References:

[1] Abraham A., Jain R., *Soft Computing in Knowledge Discovery: Methods and Applications, Saman Halgamuge and Lipo Wang (Eds.), Studies in Fuzziness and Soft Computing*, Springer Verlag Germany, Chapter 16, 20 pages, 2004.

[2] Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, pp. 1-14. Reading, MA: Addison Wesley, 1989.

[3] Ryan, C., Collins, J.J., Micheal O'Neill, . Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science: Proceedings of the First European Workshop on Genetic Programming*, pp. 83-95, 1998.

[4] Ryan, C., Micheal O'Neill, Collins, J.J. Grammatical Evolution: Solving Trigonometric Identities. *Proceedings of Mendel 1998: 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets*, pp. 111-119, 1998.

[5] Ryan, C., Micheal O'Neill,.Grammatical Evolution: A Steady State Approach. *Proceedings of the Second International*

Workshop on Frontiers in Evolutionary Algorithms, pp. 419-423, 1998.

[6] Michael O'Neill, Conor Ryan, *Grammatical evolution : evolutionary automatic programming in an arbitrary language*, Kluwer Academic Publishers, 2003.

[7] The 1998 intrusion detection off-line evaluation plan. MIT Lincoln Lab. http://www.ll.mit.edu/IST/ideval/

[8] Knowledge discovery in databases DARPA archive,http://www.kdd.ics.uci.edu/databases/kddcup99/task.html.

[9] Sabhnani M., Serpen G., "Why Machine Learning Algorithms Fail in Misuse Detection on KDD Intrusion Detection Data Set", *In Journal of Intelligent Data Analysis*, 2004.