Evolving High-speed, Easy-to-understand Network Intrusion Detection Rules with Genetic Programming

Agustin Orfila[1], Juan M. E. Tapiador[1], and Arturo Ribagorda[1]

Universidad Carlos III de Madrid, Leganes 28911, Spain
{adiaz,jestevez,arturo}@inf.uc3m.es

Abstract. An ever-present problem in intrusion detection technology is how to construct the patterns of (good, bad or anomalous) behaviour upon which an engine have to make decisions regarding the nature of the activity observed in a system. This has traditionally been one of the central areas of research in the field, and most of the solutions proposed so far have relied in one way or another upon some form of data mining–with the exception, of course, of human-constructed patterns. In this paper, we explore the use of Genetic Programming (GP) for such a purpose. Our approach is not new in some aspects, as GP has already been partially explored in the past. Here we show that GP can offer at least two advantages over other classical mechanisms: it can produce very lightweight detection rules (something of extreme importance for high-speed networks or resource-constrained applications) and the simplicity of the patterns generated allows to easily understand the semantics of the underlying attack.

1 Introduction

This paper explores the application of GP [5] to the network intrusion detection problem, particularly to automatically creating detection patterns/rules. To the best of our knowledge, the idea is somewhat recent and was proposed for the first time in 1995 by Crosbie [2]. Since then, the most relevant works have been [9, 1, 4]. The problems we have identified in these previous efforts can be summarised in two main points. First, these works do not make the most of GP. The semantics of the domain is not captured in the solutions and this situation can be improved in many aspects. Most of the times the function set chosen include primitives poorly justified, such as trigonometric functions. This makes the solutions very difficult to interpret. Second, experimental work is mainly done over the well-known benchmark of KDD-99 [3]. Although this benchmark has become a standard in the field, it has certain drawbacks such us being out of date or presenting an unrealistic high prevalence of attacks. Moreover, KDD-99 dataset was created for a contest by processing the `tcpdump` portions (i.e. raw network traffic) of the 1998 DARPA IDS Evaluation Dataset, created by Lincoln Labs under contract to DARPA [6]. The original traffic generation process and the

feature extraction done for the KDD contest remain controversial [7]. Any IDS based on data-mining is very dependant upon the dataset, since the solutions are mathematical relations between the benchmark features. As a consequence, it is unclear the behaviour exhibited by these proposals in a real scenario. In addition, a serious problem that current IDSs have to face is their application in high-speed networks or, alternatively, in resource-constrained environments. A lightweight detection is required in both cases; otherwise the IDS becomes a bottleneck (and eventually it is disconnected for performance reasons), or simply it cannot be deployed in devices with shortage of computational resources. GP, however, includes a natural mechanism to limit the amount of resources consumed by the evolved individuals (e.g. limiting the number of nodes and/or depth of trees, or associating a cost to each node in a tree and penalising trees with higher costs.) This opportunity seems to have been systematically ignored in previous works. For these reasons, instead of using KDD benchmark our work focuses on publicly available raw tcp traffic from an enterprise network [8]. All in all, it is clear that all this research supports the idea that GP can be effectively applied to intrusion detection. Nevertheless we think that further work should be done with other datasets in order to avoid the dependency of the results on the dataset. Moreover, the function set should provide more understable semantics in relation to why an event is considered an intrusion.

In this paper, we have constrained the experimental work to the capacity of GP to find analysis engines that are able to detect scanning attacks based on TCP (Transmission Control Protocol.) Therefore, traffic is only processed at the transport layer and the TCP headers are extracted to be used as inputs for our system. Results show that many TCP scans can be detected with simple rules that directly operate on the TCP header fields. Furthermore, we show that extra knowledge about the nature of the attack can be extracted thanks to the simplicity of the generated patterns and the set of operators chosen. In order to put the results in context, we compare GP solutions with those achieved by the classical machine learning algorithm C4.5 in terms of effectiveness, efficiency and semantics.

2 Design

For simplicity, in the work presented in this paper we shall restrict ourselves to analyse the capabilities of standard GP to detect a specific form of attack: scanning probes relying upon misuses of the TCP protocol. The design principles are, however, very general and can be trivially extended to other forms of network traffic and/or attacks.

For the case at hand, we extract the TCP header from each captured network packet (see Section 3 for details about the traffic source used.) The header is then processed by extracting all the fields (excluding options and checksum) and converting them into a 32-bit unsigned integer. It is important to note that each TCP flag (URG, ACK, PSH, RST, SYN, FIN) is converted separately to an attribute. The vector resulting of this conversion is the input to the analysis

engine of the IDS, which will process it according to one or more individuals obtained by GP in a training phase. Thus, the output of the engine is just 0 if the packet analysed is considered non-scanning or 1 otherwise. In what follows we describe the main GP components used in this work.

2.1 Function and Terminal sets

The set of functions define the core of solutions in standard GP. Domain knowledge is captured through the election of these functions up to a certain extent. Therefore, it is desirable to employ functions having a clear and simple semantics, as well as a fast implementation. In our case, the chosen function set comprises of logic operators (*or_logic*, *and_logic*, *not_logic*) and bitwise operators (*or*, *and*, *not*). Moreover, the function $if\_srcport\_ge\_10000$ is also introduced to provide a different point of view in the searching process, allowing to split the instances in those with a source port over or under 10000. The set of terminals is relatively simple. As we have already stated, each TCP field, including flags separately, is represented by a 32-bit unsigned integer. We also included Ephermeral Random Constants (ERCs) for completing the terminal set. An ERC is a constant value used by GP in order to generate better individuals (for a more detailed explanation on ERCs see [5].) In our system, ERCs are 32-bit random values that can be included as terminals of the IDS.

2.2 Fitness function

The chosen fitness function is the difference between the hit rate (H) and the false alarm rate (F) multiplied by the accuracy. The hit rate or detection rate stands for the probability of an alarm given an intrusion $P(A|I)$ while the false alarm rate stands for the probability of an alarm given no intrusion $P(A|NI)$. These quantities can be computed as follows:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ number\ of\ instances} \tag{1}$$

$$H = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{2}$$

$$F = \frac{False\ Positives}{False\ Positives + True\ Negatives} \tag{3}$$

2.3 Tree size limitations

We have constrained the maximum number of nodes to 20 nodes and the tree depth to 6. This was done in order to obtain short individuals with an easy interpretation of how they work. This restriction is adjustable, although in our experimentation these parameters have demonstrated to produce very small and sufficiently accurate rules.

3 Experimental work

Our experiments have been done using a modern dataset[1] [8]. This dataset corresponds to internal enterprise traffic (Lawrence Berkeley National Laboratory) recorded at a medium-sized site. It was systematically anonymised [8] and only the resulting packet header traces (available in `tcpdump/pcap` format) are publicly available. Two kinds of files for the traces are provided, anonymised separately. One corresponds to the non-scanning traffic and the other to the scanning traffic. We have used the first five traces of the dataset. Both non-scanning and scanning traffic files were filtered to get rid of non TCP traces. After this process, the number of packets under consideration is 3,415,747. Some of them were used for evolving individuals and some for the testing phase, as will be explained later. Table 1 shows the number of non-scanning and scanning TCP packets in each analysed trace after the filtering process.

Table 1. Number of TCP packets in the 5 traces of the dataset considered.

Trace number	Non-scanning	Scanning	Total
1	82,817	646	83,463
2	619,120	2,295	621,415
3	2,206,587	2,637	2,209,224
4	3,828	80	3,908
5	496,795	942	497,737
All	3,409,147	6,600	3,415,747

At a first stage, different initial populations were evolved using just one trace of the dataset. Afterwards, the best individual obtained was tested over the remaining traces. Trace number 3 has not been used for the evolution stage due to its large size (2,209,224 instances.)

For each trace, we ran 15 experiments with 15 different seeds. The results presented below correspond to the best individuals generated after these 15 experiments, with a population size of 100 individuals, a crossover probability of 0.9, and an ending condition of 25 generations. Results are compared with those obtained by the machine learning algorithm C4.5. In order to make a fair comparison, the same process was followed in the case of C4.5: one of the five traces was used to build the model and the remaining for testing purposes.

Table 2 summarises the best results obtained by GP and C4.5. It is important to note that the prevalence of scanning TCP packets in the first five traces of the dataset is 0.0019. Thus, a naive IDS stating that every TCP packet is non-scanning would achieve a 99.81% of accuracy (with H=0 and F= 1.) Therefore, when the distribution of the classes (scanning, non-scanning) is so unbalanced, accuracy is not a good measure of effectiveness. Accordingly, we show the results providing H and F. Thus, Table 2 shows that GP provides more effective rules

[1] LBNL Enterprise trace Repository, `http://www.icir.org/enterprise-tracing/`

Table 2. Summary of results

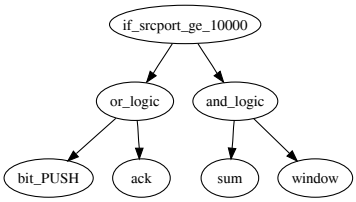
		GP $fitness = (H - F)\ Accuracy$		**C4.5**	
Trace used for evolution	Testing traces	F	H	F	H
1	2,3,4,5	0.1876	0.8013	0.0809	0.2057
2	1,3,4,5	0.0155	0.8546	0.0014	0.7936
4	1,2,3,5	0.0016	0.2979	0.0028	0.5304
5	1,2,3,4	0.0041	0.5322	0.0021	0.4231

than C4.5 for models derived from traces 1 and 5, but not for those from traces 2 and 4. The C4.5 model and the best GP individual obtained from trace 5 are represented in Figure 1.

```
=== BEST-OF-RUN ===
              generation: 13
                   nodes: 7
                   depth: 2
                    hits: 497007
TOP INDIVIDUAL:



-- #1 --
                    hits: 497007
             raw fitness: 0.9315
    standardized fitness: 0.9315
        adjusted fitness: 0.9315
    TREE:
 (if_srcport_ge_10000 (or_logic bit_PUSH ack)
                      (and_logic sum window))
```




```
urp <= 0
|   seq <= 0
|   |   srcport <= 37: 1 (119.0)
|   |   srcport > 37: 0 (31.0)
|   seq > 0
|   |   dstport <= 68
|   |   |   srcport <= 50170
|   |   |   |   dstport <= 37: 0 (425.0)
|   |   |   |   dstport > 37
|   |   |   |   |   ack <= 18602
|   |   |   |   |   |   dataoffset <= 7
|   |   |   |   |   |   |   bit_PUSH <= 0
|   |   |   |   |   |   |   |   bit_RST <= 0
|   |   |   |   |   |   |   |   |   window <= 16952
|   |   |   |   |   |   |   |   |   |   sum <= 8815: 1 (7.0)
|   |   |   |   |   |   |   |   |   |   sum > 8815: 0 (6.0)
|   |   |   |   |   |   |   |   |   window > 16952: 1 (32.0/2.0)
|   |   |   |   |   |   |   |   bit_RST > 0: 1 (9.0)
|   |   |   |   |   |   |   bit_PUSH > 0: 0 (4.0)
|   |   |   |   |   |   dataoffset > 7: 0 (8.0)
|   |   |   |   |   ack > 18602: 0 (36.0/1.0)
|   |   |   srcport > 50170: 1 (146.0)
|   |   dstport > 68
|   |   |   srcport <= 53
|   |   |   |   srcport <= 37: 0 (950.0/1.0)
|   |   |   |   srcport > 37
|   |   |   |   |   bit_PUSH <= 0
|   |   |   |   |   |   dataoffset <= 6
|   |   |   |   |   |   |   dstport <= 2953
|   |   |   |   |   |   |   |   dstport <= 1209: 1 (17.0/2.0)
|   |   |   |   |   |   |   |   dstport > 1209
|   |   |   |   |   |   |   |   |   dstport <= 1367: 0 (10.0)
|   |   |   |   |   |   |   |   |   dstport > 1367
|   |   |   |   |   |   |   |   |   |   dstport <= 2379: 1 (14.0/1.0)
|   |   |   |   |   |   |   |   |   |   dstport > 2379: 0 (4.0)
|   |   |   |   |   |   |   dstport > 2953: 1 (13.0)
|   |   |   |   |   |   dataoffset > 6: 0 (9.0)
|   |   |   |   |   bit_PUSH > 0: 0 (22.0)
|   |   |   srcport > 53
|   |   |   |   bit_RST <= 0: 0 (494629.0/2.0)
|   |   |   |   bit_RST > 0
|   |   |   |   |   srcport <= 137
|   |   |   |   |   |   srcport <= 89: 0 (16.0)
|   |   |   |   |   |   srcport > 89
|   |   |   |   |   |   |   ack <= 34845: 1 (6.0)
|   |   |   |   |   |   |   ack > 34845: 0 (3.0)
|   |   |   |   |   srcport > 137: 0 (460.0/1.0)
urp > 0
|   srcport <= 3967
|   |   dstport <= 3875
|   |   |   sum <= 20241
|   |   |   |   urp <= 26494: 0 (5.0)
|   |   |   |   urp > 26494: 1 (48.0)
|   |   |   sum > 20241: 1 (486.0/1.0)
|   |   dstport > 3875
|   |   |   dstport <= 61951
|   |   |   |   srcport <= 137
|   |   |   |   |   seq <= 7913: 1 (3.0)
|   |   |   |   |   seq > 7913: 0 (2.0)
|   |   |   |   srcport > 137: 0 (95.0)
|   |   |   dstport > 61951: 1 (43.0)
|   srcport > 3967: 0 (79.0)
```

Fig. 1. Comparison of GP top individual (left) and C4.5 model (right) obtained from trace 5

As can be seen, GP solution is much simpler than C4.5 one (7 vs. 63 nodes.) Thus, it is easy to understand why the generated GP rule detects a scan. In contrast, the C4.5 tree is complex enough (63 nodes) to make it difficult to comprehend the nature of the detection. This is clearly an advantage in favour of GP technique.

Apart from semantics reasons, the complexity of the rules has an impact on the detection efficiency. Table 3 shows a comparison in terms of the number of basic operations per TCP packet required for the worst case. The *and*, *or*, *not*, *and_logic*, *or_logic*, *not_logic* functions and load operations are considered basic (i.e. with a cost of 1 unit), while *comparisons* are assumed to be formed by two basic operations. GP models are generally much more efficient than C4.5 ones (the only exception corresponds to trace 4 due to its small size–see Table 1.)

Table 3. Efficiency of GP vs. C4.5: Number of basic operations per TCP packet.

Trace used for evolution	**GP** $fitness = (H - F)\ Accuracy$	**C4.5**
1	12	28
2	18	33
4	11	4
5	7	40

References

1. A. Abraham, C. Grosan, and C. Martin-Vide. Evolutionary design of intrusion detection programs. *International Journal of Network Security*, 4(3):328–339, 2007.
2. M. Crosbie and E. H. Spafford. Applying genetic programming to intrusion detection. In *Working Notes for the AAAI Symposium on GP*, 1995.
3. C. Elkan. Results of the KDD'99 classifier learning contest. `http://www-cse.ucsd.edu/users/elkan/clresults.html`, September 1999.
4. K. Faraoun and A. Boukelif. Genetic programming approach for multi-category pattern classification applied to network intrusions detection. *The International Arab Journal of Information Technology*, 4(3):237–246, 2007.
5. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
6. R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.
7. Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *Proceedings of the 6th RAID*, volume 2820 of *LNCS*, pages 220–237, 2003.
8. R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet measurement. IMC '05*, pages 1–14, New York, NY, USA, 2005. ACM.
9. D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, June 2005.