

Policy Evolution with Grammatical Evolution

Yow Tzu Lim*, Pau Chen Cheng†, John Andrew Clark* and Pankaj Rohatgi†

*Department of Computer Science, University of York
York, North Yorkshire, United Kingdom

Email: {yowtzu, jac}@cs.york.ac.uk

†Department of Security and Privacy, IBM Thomas J. Watson Research Center
Hawthorne, NY, United States

Email: {pau, rohatgi}@us.ibm.com

Abstract—Security policies are becoming more sophisticated. Operational forces will often be faced with making tricky risk decisions and policies must be flexible enough to allow appropriate actions to be facilitated. Access requests are no longer simple subject access object matters. There is often a great deal of context to be taken into account. Most security work is couched in terms of risk management, but the benefits of actions will need to be taken into account too. In some cases it may not be clear what the policy should be. People are often better at dealing with specific examples than producing general rules. In this paper we investigate the use of Grammatical Evolution (GE), a novel evolutionary algorithm to attempt to infer Fuzzy MLS policy model from decision examples. This approach couches policy inference as a search over a policy space for a policy that is most consistent with the supplied training decision set. Each candidate policy considered is associated with a fitness that indicates how consistent is the policy with the decisions provided. This fitness is then optimised using GE techniques. The results show this approach is promising.

I. INTRODUCTION

In computer systems, a security policy is essentially a set of rules specifying the way to secure a system for the present and the *future*. Forming a security policy is a challenging task: the system may be inherently complex with many potentially conflicting factors. Traditionally security policies have had a strong tendency to encode a static view of risk and how it should be managed (most typically in a pessimistic or conservative way) [1]. Such an approach will not suffice for many dynamic systems which operate in highly uncertain, inherently risky environments. In many military operations, for example, we cannot expect to predict all possible situations.

Much security work is couched in terms of risk but in the real world there are benefits to be had. In military operations you may be prepared to risk a compromise of confidentiality if not doing so could cost lives. There is a need for operational flexibility in decision making, yet we cannot allow recklessness. Decisions need to be defensible and so must be made on some principled basis. People are typically better at making

specific decisions than in providing abstract justification for their decisions. It is very useful to be able to codify in what a “principled basis” consists since this serves to document “good practice” and facilitates its propagation.

The above discussion has been couched in terms of human decision making. In some environments the required speed of system response may force an automated decision. Such automated decisions must also be made on a “principled basis”, and some of these decisions may be very tricky. Automated support must be provided with decision strategies or rules to apply.

In this paper we investigate how security policy rules can be extracted automatically from examples of decisions made in specified circumstances. This is an exercise in *policy inference*. The automation aspect of the inference is doubly useful: automated inference techniques can discover rules that humans would miss; and policies can be dynamically inferred as new examples of tricky decisions become available. Thus the current policy can evolve to reflect the experience of the system.

For example, if a human determines what the proper response should be based upon the information available, either in real-time or post facto, a conclusion is drawn that similar responses should be given under similar circumstances. Essentially, we attempt to partition the decision space such that each partition is associated with a response that is commensurate with the risk vs. benefit trade-off.

In practice, different decision makers may come to different decisions in the same circumstances, particularly if the decisions are tricky. Decision makers may use data that is not available to the inference engine to reach a decision, or else one decision-maker may simply have a different appetite for risk. Any inference technique must be able to handle sets of decision examples that do not seem to be entirely consistent. The chosen inference approach is Grammatical Evolution (GE).

The organisation of this paper is as follows: Section II discusses the related work. Section III introduces Grammatical Evolution. Section IV presents the Fuzzy MLS policy model, which serves as the learning target throughout the paper. Sections V and VI present the experimental setup and results. Finally Section VII concludes the report with a summary of the experiment results and possible future work.

Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

II. RELATED WORK

The application of machine learning techniques on security domain has been focused on intrusion detection and anomaly detection. Machine learning techniques are used to discover useful patterns of system features that describe the system behaviour. Then, these features are used to recognise anomaly and intrusion detection. For detail, refer [2]. However, there are limited known attempts in generating the policy automatically from previous decision examples using machine learning techniques. In [3] autonomic security policies were mentioned yet no results have appeared. The only published works are found in [4] in which Genetic Programming is used to infer various Fuzzy MLS policy model from decision examples. The results show GP based approach is promising.

On the other hand, rule inferencing techniques have been around for many years in machine learning domain. Emphasis in this paper is placed on Grammatical Evolution (GE). GE has emerged in recent years as a promising technique. A list of applications and other GE papers can be found at <http://www.grammatical-evolution.org/pubs.html>. In [5], [6] GE is used to learn investment strategy (trading rules) using the market stock indices. The results show the investment strategy inferred yields profitable performance for the trading periods analysed. In [7] GE is used to evolve rules for foreign exchanges. The rules learnt outperforms the benchmark buy and hold strategy over all the trading periods analysed. However, there have been no previous known research on GE in the our domain of interest — security policy.

III. GRAMMATICAL EVOLUTION

Evolutionary algorithms (EAs) are problem solving techniques inspired from natural selection¹. An initial population of individuals, which represent candidate solutions to the problem in question, are generated. There are a variety of ways of doing this but some form of randomised approach is normally used. Each individual can be associated with a “fitness” that measures how well each candidate solves the problem. In many applications this is formed by the application of some defined function, $f(sol)$ to the candidate solution, sol . In others, the “real world” acts as the cost function. For example, it is far easier to see how much power a program consumes by running the program and measuring its power consumption than it is to derive and use a predictive model of power consumption.

The fitness of the initial population are evaluated and the population is then subject repeatedly to the three evolutionary operators:

- Selection: individuals are selected for breeding according to fitness (natural selection);
- Crossover: selected individuals are bred; parts of each solution are exchanged to form two new individuals;
- Mutation: elements within a solution are perturbed in some way. This serves to diversify solution elements in

¹Natural selection states that individual that are best adapted to the environment have better chance to survive and reproduce for next generation.

the population. Given an initial population, repeated application of selection and crossover alone might otherwise not be able to reach parts of the search space.

This evolutionary process produces populations of individuals (candidate solutions) that are increasingly better suited to the environment (problem). Commonly, the stopping criterion for EAs is either that a “good enough” solution has been found or that a preset number of generations have been produced.

There are a variety of EAs. Representations of solutions vary a lot. In most applications solutions are represented as linear lists of elements of the solution. Genetic Programming has traditionally used tree representations. The nodes in the tree can be classified into two groups: the terminal set T and function set F. The terminal set T consists of constants and variables (the leaf nodes) and the function set F consists of functions, operators and statements (the non-leaf nodes). Two standard evolutionary operators in GP: *crossover* and *mutation* in GP are defined as follows:

- 1) Crossover is performed on two trees. A sub-tree in each tree is chosen randomly and swapped with the other.
- 2) Mutation is performed on one tree. A node in the tree is chosen randomly and replaced with a new node or sub-tree randomly. Mutation helps to introduce diversity into the population.

However, in this standard form, there is no way to ensure the all the functions in each tree to have appropriate data types (children). In [8], Montana introduces Strongly Typed Genetic Programming (STGP). STGP augments each node with a type and a set of type rules is defined to specify how nodes in a tree can be connected with one another. For example, we may require that \geq take two floating point children and return a boolean value. The population initialisation and genetic operations must obey the type rules. In STGP only well-typed individuals are maintained in the population. An alternative way to this is to employ grammar to control the structure of the tree.

These run-time type or grammar conformance checking on each individual is expensive. Instead, Grammatical Evolution (GE) uses variable length binary strings individuals which serve as indices that map a set BNF language grammar rules to programs. These programs can then be executed for fitness evaluation purposes. To solve a problem using GE, the language grammar must first be specified in Backus Naur Form (BNF). The grammar of a language in BNF is represented as a tuple $\{N, T, S, P\}$ [9] where N is the non-terminal set, T is the terminal, S is a special element of N called start symbol which is expanded first. P is the set of production rules (also called derivation rules) that is used to guide the expansion from N to T . A production rule has the following format:

```
<non-terminal> ::= <expr-of-symbols-1>
                  | <expr-of-symbols-2>
                  | ...
```

This rule states that the non-terminal symbol on the left hand side of the rule is to be substituted by one of symbol

expressions on the right hand side of the rule. | is used to indicate the choice of expressions. An example of BNF grammar is shown as follows [9]:

$$\begin{aligned} N &= \{ \langle \text{expr} \rangle, \langle \text{op} \rangle, \langle \text{pre_op} \rangle \} \\ T &= \{ +, -, *, /, X, Y, (,) \} \\ S &= \langle \text{expr} \rangle \end{aligned}$$

P consists of a set of production rules as follows:

$$\begin{aligned} \langle \text{expr} \rangle &::= \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle & (0) \\ &| (\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle) & (1) \\ &| \langle \text{var} \rangle & (2) \\ \langle \text{op} \rangle &::= + & (0) \\ &| - & (1) \\ &| / & (2) \\ &| * & (3) \\ \langle \text{var} \rangle &::= X & (0) \\ &| Y & (1) \end{aligned}$$

This grammar defines a set of arithmetic expressions over the variables X and Y . Although there is no distinction made on the BNF grammar between the functions and terminals in GE, this distinction is still required in the implementation level [9].

A. Genotype-phenotype mapping

The mapping process between the variable length binary individuals and programs using the BNF grammar is known as genotype-phenotype mapping. This is best illustrated with an example. In each individual (genome), every 8 consecutive bits is viewed as an integer which are more commonly known as codon. Consider an individual with the following codons:

$$[10, 5, 51, 8, 16, 49, 30, 18]$$

As mentioned, the mapping process begins with the start symbol. The codon value is read to determine the usage of a certain production rule to expand the symbol using a modulus mapping function as follows:

$$\begin{aligned} \text{Rule} &= \text{Codon value} \% \\ &\quad \text{Total number of production rules} \end{aligned}$$

where $\%$ is the arithmetic modulus operator. Assuming the start symbol is $\langle \text{expr} \rangle$, as the first codon value is 10 and there are 3 production rules for $\langle \text{expr} \rangle$, therefore $10 \% 3 = 1$, the production rule 1 is used to replace $\langle \text{expr} \rangle$ with $(\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle)$.

Assuming leftmost derivation is used, we will now expand the left most $\langle \text{expr} \rangle$ after (which is a terminal. The second codon value is used to select the production rules. As $5 \text{ mod } 3 = 2$, the production rule 2 is selected. The leftmost $\langle \text{expr} \rangle$ is replaced by $\langle \text{var} \rangle$ resulting $(\langle \text{var} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle)$. The next non-terminal to be expanded is $\langle \text{var} \rangle$. As the next codon is 51 and the number of rule choices available is 2, $\langle \text{var} \rangle$ with Y .

The process continues until a complete program is generated, when all the non-terminals have been transformed into terminals. While this is desirable, some problems can arise during the mapping process. Firstly, there is a possibility that the codon values may run out before the entire mapping process is complete. To overcome this problem, individual can be wrapped around to reuse the codon value, i.e., the reading of the last codon (18 in the example) will be followed by the first codon (10 in the example).

The mapping process also can be indefinitely long if the context free grammar used is recursive. Consider the production rules for $\langle \text{expr} \rangle$. If the codon value always maps to the production rule 0 or 1, the mapping process is never ending. The $\langle \text{expr} \rangle$ is replaced by either $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ or $(\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle)$. In both case, the next leftmost non-terminal is again $\langle \text{expr} \rangle$ itself. This problem can be resolved by setting an upper bound on the number of times the production rules that can be performed. In the original algorithm, this is achieved indirectly by setting a limit on the number of times the individual can be wraparound and thus the number of codons available. If a complete program is not fully generated (the non-terminals have not been transformed to terminals) when the wrapped around threshold is reached, this individual is given the lowest possible fitness value to decrease the probability of this individual to be selected in selection step.

The greatest advantage of GE is that it allows artifacts to be evolved in a grammar compliant way and many solution spaces can be defined using grammars. The use of grammar also makes program generation in arbitrary languages possible by only changing the BNF grammar. The search algorithm is also an independent component in the system. Instead of using the standard genetic algorithm, it is possible to use other search. For example, Grammatical Swarm [10] uses Particle Swarm Optimisation (PSO) to carry out the search. This feature allows GE to reap the advantages of any improvement in any evolutionary algorithm.

IV. FUZZY MLS MODEL

Fuzzy MLS policy model [11] is an adaptive extension to traditional MLS (multi-level security) Bell-LaPadula policy model [12]. In the Bell-LaPadula policy model, every subject and object is assigned a security label (\langle sensitivity level, categories set \rangle). For a read access, r , the policy can be summarised as follows:

$$\begin{aligned} \text{IF } sl \geq ol \text{ AND } sc \supseteq oc \text{ THEN } r \text{ is allowed} \\ \text{IF } sl < ol \text{ OR } sc \not\supseteq oc \text{ THEN } r \text{ is denied} \end{aligned} \quad (1)$$

where sl and ol are subject and object sensitivity levels and sc and oc are subject and object category sets. In other words, a subject can access an object iff the subject is trustworthy enough ($sl \geq ol$) and has the legitimate “need-to-know” ($sc \supseteq oc$) to access the object. In terms of risk, the traditional MLS policy can be viewed as setting a fixed trade-off between risk of information disclosure versus the benefit an organisation can gain from it. As indicated in [11], it is a non-adaptive, binary

access control decision model where accesses have been pre-classified as having either acceptable or unacceptable risk and only accesses with acceptable risk are allowed.

The Fuzzy MLS model uses this risk based rationale, but extends the MLS model to be based on risk management rather than risk avoidance inherent in the binary decision process [1]. The Fuzzy MLS model takes a more flexible and sophisticated view by computing *quantified estimates* of the risk from *unauthorised disclosure of information* and using these estimates to build a risk scale shown in Figure 1. The risk

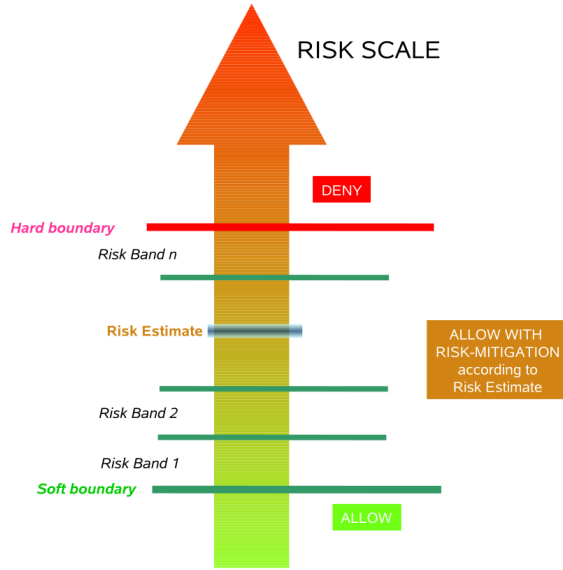


Fig. 1. Risk adaptive access control on a risk scale

scale is divided into multiple bands. Each band is associated with a decision. The risk in the bottom band is considered low enough so the decision is simply *allow* whereas the risk in the top band is considered too high so the decision is *deny*. Each band between the top and bottom is associated with a decision *allow with band-specific risk mitigation measures*.

The Fuzzy MLS model defines risk as the *expected value of damage* caused by unauthorised disclosure of information:

$$risk = (expected\ value\ of\ damage) \times (probability\ of\ unauthorised\ disclosure) \quad (2)$$

The value of the damage is estimated from the object's sensitivity level. The probability of unauthorised disclosure is estimated by quantifying two "gaps": one between the subject's and the object's sensitivity levels and the other between the subject's and the object's category sets. For simplicity, this experiment looks only at the sensitivity levels and assumes the categories sets are the same², thus risk becomes a function of subject and object sensitivity levels only. For more detail on risk quantification, see [11].

²Therefore the gap between categories sets is 0.

In [4], a way to partition the risk bands is defined to map an estimated risk to a risk band:

$$band(risk(sl, ol)) = \min(\lfloor \log_{10}(risk(sl, ol)) \rfloor, N - 1) \quad (3)$$

The band numbers start from 0, N is the number of bands desired on the scale and the function $risk(sl, ol)$ is defined in Appendix according to [11]. The intuition of using base-10 logarithm in (3) is to use a risk band number to represent the *order of magnitude* of risk. Since each band is associated with a decision, a risk band number computed using (3) represents a possible decision in the policy.

V. EXPERIMENTS

A policy can be viewed as a function which maps decision making factors to a decision. For example, the "no read up" part of traditional MLS Bell-LaPadula model can be viewed as a boolean function $access(sl, ol, sc, oc)$ which maps to *True* iff $sl \geq ol$ and $sc \supseteq oc$. In the Fuzzy MLS model, this mapping is described as a composition of the band function and risk function in (3).

In this experiment, standard GE using a steady state genetic algorithm as the search algorithm is used to search for an equivalent function of this composition function using a set of decision examples.

A. Grammar

Each individual in the population corresponds to a candidate function that corresponds to a policy. The BNF grammar that describes the structure of each individual is as follows:

$$\begin{aligned} N &= \{ \langle \text{expr} \rangle, \langle \text{sub_expr} \rangle, \langle \text{unary_op} \rangle, \\ &\quad \langle \text{binary_op} \rangle, \langle \text{var} \rangle, \langle \text{const} \rangle, \langle \text{digit} \rangle \} \\ T &= \{ sl, ol, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \} \\ S &= \langle \text{expr} \rangle \end{aligned}$$

and P consists of a set of production rules as follows:

$$\begin{aligned} \langle \text{expr} \rangle & ::= \langle \text{unary_op} \rangle (\langle \text{expr} \rangle) \\ & \quad | \langle \text{binary_op} \rangle (\langle \text{expr} \rangle, \langle \text{expr} \rangle) \\ & \quad | \langle \text{sub_expr} \rangle \\ \langle \text{sub_expr} \rangle & ::= \langle \text{var} \rangle \\ & \quad | \langle \text{const} \rangle \\ \langle \text{unary_op} \rangle & ::= \sin \\ & \quad | \cos \\ & \quad | \exp \\ & \quad | \text{protectedLog10} \\ & \quad | \text{ceil} \\ & \quad | \text{floor} \\ & \quad | \exp \\ & \quad | - \\ \langle \text{binary_op} \rangle & ::= \text{add} \\ & \quad | \text{minus} \\ & \quad | \text{multiply} \\ & \quad | \text{protectedDiv} \\ & \quad | \text{min} \\ & \quad | \text{max} \\ \langle \text{var} \rangle & ::= sl \mid ol \end{aligned}$$

```

<const>      ::= 0.<digit><digit>
<digit>      ::= 0|1|2|3|4|5|6|7|8|9

```

The primitive operators are wrapped as a function call to prevent any bias being introduced among the operators. All functions work in accordance to the specification defined in the ANSI C standard `<math.h>` library except four operators as follows: $\min(x, y)$, $\max(x, y)$, $\text{protectedDiv}(x, y)$ and $\text{protectedLog10}(x)$. \min and \max operators return the minimum and maximum values between x and y , $\text{protectedDiv}(x, y)$ returns x / y if $y \neq 0$ and 1 if $y = 0$, and $\text{protectedLog10}(x)$ returns $\log_{10}(x)$ if $x > 0$ and 1 otherwise.

Instead of using ephemeral random constant (ERC)³, the `const` and `digit` rules are used to generate random constant in the range of $(-1, 1)$. Generating random constant in this fashion enables random number to partake in the evolutionary process.

B. Evolutionary Operators

At each iteration step, the crossover and mutation operations are applied to individual with the probabilities of $P_{\text{crossover}}$ and P_{mutation} respectively. $P_{\text{crossover}} = 0.9$ and $P_{\text{mutation}} = 0.01$ are used in all experiments. The mutation operator used is the standard bit-level GE mutation which flip each bit of the individual with P_{mutation} . Two experiments have been carried out to investigate two different implementation of crossover operations; one-point crossover and effective crossover. One point crossover first chooses a point randomly on each of the two selected individuals and all data beyond that point is swapped between the two individuals. The chosen crossover point may happen to be after the effective length of the individual (the portion of the individual that are actually used to select the rules) and render the crossover operation ineffective. Effective crossover restricts the chosen crossover point in the range of effective length.

The steady state genetic algorithm is used in as the search algorithm to evolve the linear individual with of 1% replacement rate, i.e., the percentage of the population that are replaced at each iteration. Roulette Wheel Selection Scheme is used as the selection scheme in which the probability for an individual to be selected is directly proportional to its fitness.

C. Initial Population

Two experiments have been carried out, each with a initial population with the size of 1024. The population in the first experiment is initialised with linear individual of variable lengths in the range of $[\text{Length}_{\min}, \text{Length}_{\max}]$. $\text{Length}_{\min} = 15$ and $\text{Length}_{\max} = 25$ are used in all experiments.

The sensible initialisation method [13] is used in the second experiment. This initialisation method models the popular *ramp half and half* initialisation method [14]; it takes the population size, minimum and maximum heights of the trees permitted in the population as input parameters and generates approximately 50% full trees with maximum height while the

50% have heights between the minimum and the maximum heights.

The sensible initialisation works as following: First, each rule in the grammar is assigned with two properties; the minimum number of mapping required for the non-terminal on the left hand side of the rule to be completely mapped to terminals and whether the rule is recursive. As each derivative tree is built, only rules that can fit the remaining depth of the tree is selected. To generate full trees, the recursive rules is always chosen whenever possible. To generate trees with variable heights between minimum and maximum heights, the recursive and non-recursive rules are chosen with equal probabilities. The final step in the initialization is to map the nodes of the tree reversely into corresponding codons. The minimum and maximum heights of the derivative tree are set to be 1 and 10 in all experiments.

D. Fitness evaluation

The fitness evaluation for each individual is evaluated against a set of training examples. Since only the sensitivity level aspect is considered, each example x is a $(sl_x, ol_x, band_x)$ triple, where $band_x$ is calculated using (3) and therefore all the examples used are correct. The training set used consists of all possible 100 (sl, ol) integer pairs for sl and ol in $[0, 9]$.

Two experiments have been carried out, each uses a different fitness function. In the first experiment, the weighted fitness function (4) presented in [15] is used. This allows us to compare the performances of GE and GP side by side.

Two principles are used to determine the score for a decision made by an individual. For an example x and an individual i , if i evaluates x to be in band j (j is $i(x)$ rounded to the nearest integer), then:

- For a correct decision, *reward more* the higher the risk band; i.e., reward higher j more than lower j . (We care more about higher risk bands)
- For an incorrect decision, *punish more* the more the decision deviates from the target; i.e., punish more as $|j - band_x|$ becomes larger. Also, punish *under-estimation* of the risk band more than over-estimation of it; i.e., punish more if $band_x > j$.

Based upon these principles, the fitness of an individual i , $fitness(i)$ is defined as follow:

$$fitness(i) = \sum_{\forall \text{ example } x} score(x) \quad (4)$$

where

$$score(x) = \begin{cases} band_x + 1 & \text{if } j \equiv band_x, \\ -(band_x - j) & \text{if } band_x > j, \\ -(j - band_x)/2 & \text{if } band_x < j \end{cases}$$

The fitness function in the second experiment is inversely proportional to the sum of absolute differences between the value an individual evaluated to with the correct band encoded

³An ERC is a number in which its value is initialised randomly and remains constant once initialised

in the examples. Formally, the fitness of an individual i , $fitness(i)$ is defined as follow:

$$fitness(i) = \frac{1}{1 + \sum_{\forall \text{ example } x} |round(eval(i)) - band_x|} \quad (5)$$

The sum of absolute differences is added with 1 before inversion to avoid division by zero and therefore the fitness of an perfect individual now becomes 1.

E. Policy resolution

The learnt function might not be perfect; sometimes the function might map a particular (sl, ol) pair to a value that is out of band range. When this happens, we assume that all out-of-range values represent the highest band, i.e., in our case, any value not in the range $[0, 9]$ is assumed to be 9. This is consistent with the usual attitude to security which favours overestimation of risk rather than underestimation. In a future run-time deployment of our inference approach it would be possible to involve human interaction. An alert would be given to the security administrator and the administrator would decide which band an input should be mapped to. Then, this decision can be used as a new training example.

F. Experimental Setup Summary

This experiment is carried out using the libGE-0.26 and GALib v2.4.7 with all the parameter settings remain to be default values unless specified otherwise. The experimental setup is summarised in Table I.

Objective	Search for the nearest equivalent function of $band(risk(sl, ol))$ in (3)
Terminal Operands	$\{sl, ol, r -1.0 < r < 1.0\}$
Unary Operators	$\{sin, cos, exp, protectedLog10, ceil, floor, exp, -\}$
Binary Operators	$\{add, minus, multiple, protectedDiv, min, max\}$
Fitness function	Two fitness functions are investigated as follows: 1) Weighted fitness function in (4) 2) Sum of absolute differences in (5)
Population initialisation	Two initialisation methods are investigated as follows: 1) Random initialisation — individuals with variable lengths between $[15, 25]$ (default) 2) Sensible initialisation — individuals map grammar rules to derivative trees with maximum height of 10 (default)
Evolutionary Operators (P)	Two different sets of operators are investigated as follows: 1) One-point crossover (0.9) (default), bit-level mutation (0.01) (default) 2) Effective crossover (0.9) (default), bit-level mutation (0.01) (default)
Generation	500
Population size	1024

TABLE I
EXPERIMENTAL SETUP SUMMARY FOR FUZZY MLS POLICY INFERENCE

VI. RESULT AND DISCUSSION

Each experiment described is run 10 times. Two testing sets are used to evaluate the performance of the best individual. The first testing set is same as the first 100-example training set. This testing set provides a good indication on how much “knowledge” has been acquired by the approach employed in a fixed number of generations. The second testing set consists of 100 randomly generated (sl, ol) pairs where sl and ol are real numbers in $[0.0, 9.0]$. Therefore, most of these examples are unseen yet similar to training examples. This set provides a good measure on how much the acquired knowledge can be applied for unseen cases. The performance in terms of the sum of differences between the band of the best individual of each generation evaluated to and the band encoded in each of the 100 examples in the training set is shown in Figure 2 and 3 respectively.

The results show that the performance of the individuals evolved using sum of absolute differences outperform the ones evolved using sum fitness function in both cases at all time during evolution. The sum of absolute differences fitness function provides faster learning speed (steeper slope); it only requires ≈ 100 generations to has the population to become stable as opposed to ≈ 250 generations required by the weighted fitness function. However, the uses of effective crossover and sensible initialisation do not provide any performance gain.

In comparison to the results obtained using Genetic Programming with the same weighted fitness function and similar parameter settings [15], the performance of GE is superior over GP in terms number of correct bands and mean difference from target band in all cases. The results are summarised in Table II. The result comparison using sum of absolute differences fitness function cannot be done because there is no equivalent experiment carried out in GP.

Experiment	Testing set	Difference from target band				Mean difference
		0	1	2	≥ 3	
GP	1	62.3	9.6	6.0	22.1	1.248
	2	43.6	20.7	7.3	28.4	1.703
GE (SI ¹ , EC ³)	1	78.0	11.8	4.1	6.1	0.435
	2	47.7	39.5	7.3	5.5	0.766
GE (SI ¹ , OC ⁴)	1	80.5	10.6	3.4	5.5	0.373
	2	51.4	37.0	7.0	4.6	0.672
GE (RI ² , EC ³)	1	78.7	10.7	4.2	6.4	0.456
	2	49.8	38.4	6.7	5.1	0.731
GE (RI ² , OC ⁴)	1	76.6	11.6	4.4	7.4	0.504
	2	47.0	38.6	7.5	6.9	0.843

¹Sensible Population Initialisation ³Effective Crossover
²Random Population Initialisation ⁴One-point Crossover

TABLE II
PERFORMANCE COMPARISON BETWEEN POLICIES INFERRED USING GP AND GE WITH WEIGHTED FITNESS FUNCTION (4).

Furthermore, the size of the best individual (which is essentially a derivative tree) found in GE is much more smaller compared to the one found in GP (in terms of the number nodes). This can be explained by the fact that conforming to a grammar is relatively more difficult than conforming to the type correctness imposed in GP. One example of the best

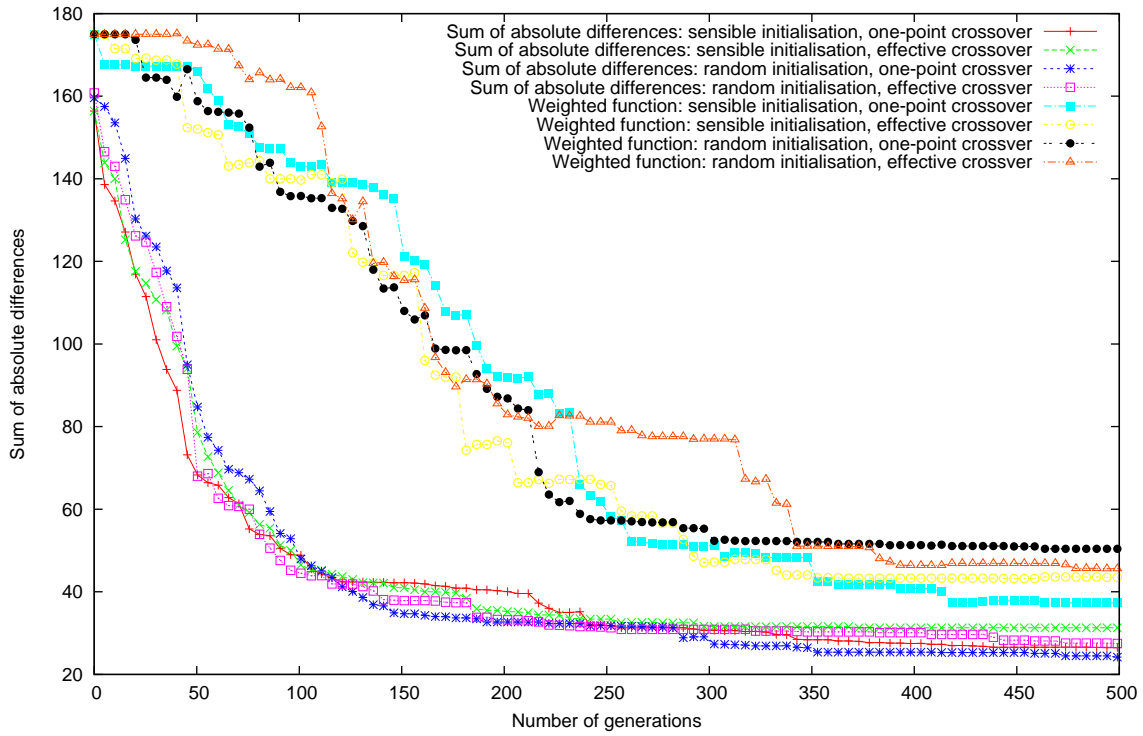


Fig. 2. The average over 10 runs of the sum of absolute differences between the band of best individual in each generation evaluated to and the band encoded in each of the 100 examples in Testing Set 1

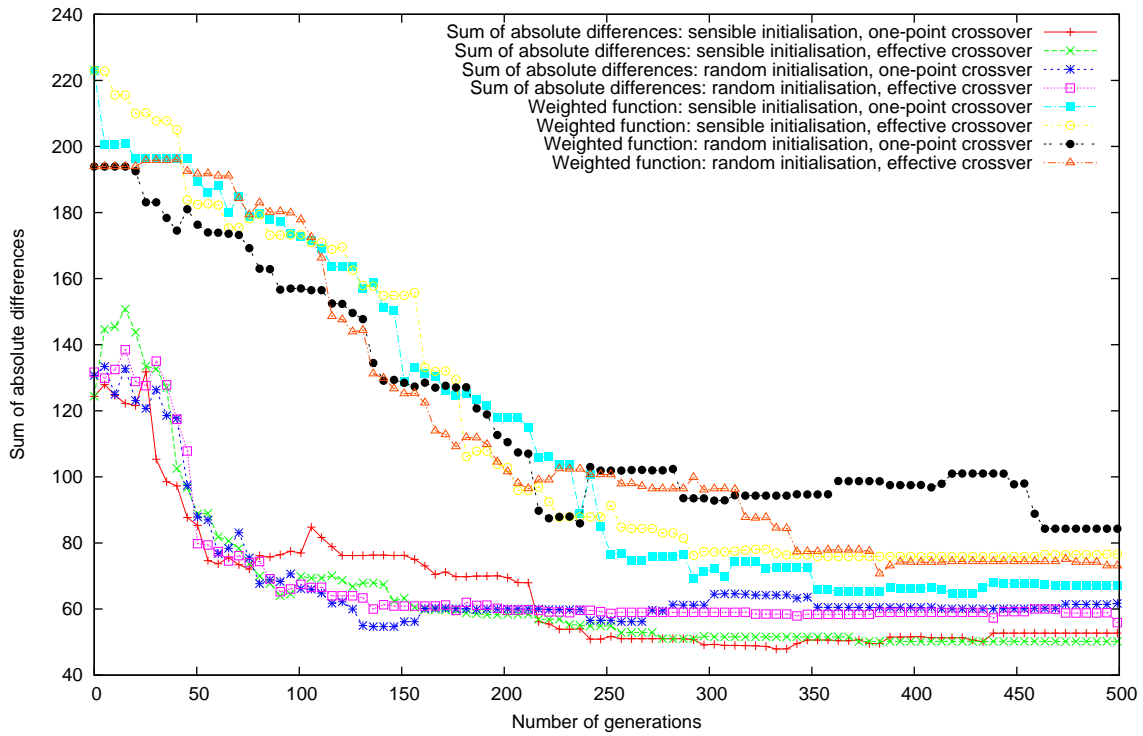


Fig. 3. The average over 10 runs of the sum of absolute differences between the band of best individual in each generation evaluated to and the band encoded in each of the 100 examples in Testing Set 2

individuals is $\min \left(ol, ol * \frac{-ol}{e^{sl} * \frac{0.87/0.03}{[0.97 - [-sl]] + ol - e^{ol}}} \right)$. The average distance between the predicted band by this individual and the target band is only 0.100 band in Training Set 1 and 0.390 band in Training Set 2.

VII. CONCLUSION

In this paper we investigate the possibility of using Grammatical Evolution, a grammar based evolutionary algorithm to infer Fuzzy MLS policy from examples. The results show that this approach is promising. Also, the policies inferred using GE are found to outperform the ones inferred using GP with similar settings [15]. While the sum of absolute differences fitness function is found to outperform the weighted fitness function, the uses of sensible initialisation and effective crossover in experiment do not provide any performance gain.

Inferring security policy with evolutionary algorithms is a very interesting domain. Some envisaged work includes: the use of a “less ideal” training set, possibly with the inclusion of wrong examples and skewed distribution on the examples set. The fuzzy set membership function concept can also be incorporated by considering each decision as a fuzzy set and the search target becomes the fuzzy set membership function for each class. This approach is found to be able to increase the accuracy performance and more resilient to missing examples in the training set [15].

Our GE based inference approach has the potential to make a significant contribution to the field of policy inference. Everyone accepts that policy specification is currently hard, and things are set to worsen as systems are deployed in ever more complex environments with increasing sophistication and subtlety of decision making needed. The work reported here shows that the technique has very considerable promise.

APPENDIX A: RISK COMPUTATION

In [11], the risk resulted from the “gap” between a subject’s and an object’s sensitivity levels (sl and ol) is estimated using the following formula:

$$risk(sl, ol) = Val(ol) \times P_1(sl, ol) \quad (6)$$

$Val(ol)$ is the estimate value of damage and is defined in [11] as

$$Val(ol) = a^{ol}, \quad a > 1$$

The object sensitivity level is considered to be the *order of magnitude* of damage and hence $Val(ol)$ is defined as an exponential formula. In our experiments we set a to be 10. $P_1(sl, ol)$ is the probability of unauthorised disclosure and is defined in [11] as a sigmoid function:

$$P_1(sl, ol) = \frac{1}{1 + \exp(-k(TI(sl, ol) - mid))}$$

$TI(sl, ol)$ is called the *temptation index* which indicates how much the subject with sensitivity sl is tempted to leak information with sensitivity level ol ; it is defined as:

$$TI(sl, ol) = \frac{a^{(ol-sl)}}{M - ol}$$

The intuition for $P_1(sl, ol)$ and $TI(sl, ol)$ can be found in [11]. The value mid is the value of TI that makes P_1 equal 0.5; the value k controls the slope of P_1 . The value M is the *ultimate object sensitivity* and the temptation TI approaches infinity as ol approaches M ; the intuition is that access to an object that is as sensitive as or more sensitive than M should be controlled by human beings and not machines. In our experiments, the maximum value for sl and ol is 10; the settings for k , mid and M are $k = 3$, $mid = 4$, $M = 11$.

REFERENCES

- [1] “Horizontal Integration: Broader Access Models for Realizing Information Dominance,” The MITRE Corporation JASON Program Office, Mclean, Virginia, Tech. Rep. JSR-04-132, Dec 2004.
- [2] D. Barbara, *Applications of Data Mining in Computer Security*, S. Jajodia, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2002.
- [3] P. D. McDaniel, “Policy Evolution: Autonomic Environmental Security,” December 2004. [Online]. Available: www.patrickmcdaniel.org/talks/serc-12-04.pdf
- [4] Y. T. Lim, P. C. Cheng, J. A. Clark, and P. Rohatgi, “Policy Evolution with Genetic Programming,” IBM Research Report RC24442, Tech. Rep., 2008.
- [5] I. Dempsey, M. O’Neill, and A. Brabazon, “Adaptive Trading with Grammatical Evolution,” in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. Vancouver: IEEE Press, 6-21 Jul. 2006, pp. 9137–9142.
- [6] T. Brabazon, M. O’Neill, C. Ryan, and J. J. Collins, “Uncovering Technical Trading Rules Using Evolutionary Automatic Programming,” in *Proceedings of 2001 AAANZ Conference (Accounting Association of Australia and NZ)*, Auckland, New Zealand, 1-3 Jul. 2001.
- [7] T. Brabazon and M. O’Neill, “Trading Foreign Exchange Markets Using Evolutionary Automatic Programming,” in *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, A. M. Barry, Ed. New York: AAAI, 8 Jul. 2002, pp. 133–136. [Online]. Available: <http://www.grammatical-evolution.org/geccos2002/brabazon.ps>
- [8] D. J. Montana, “Strongly typed genetic programming,” *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.
- [9] M. O’Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, ser. Genetic programming. Kluwer Academic Publishers, 2003, vol. 4. [Online]. Available: <http://www.wkap.nl/prod/b/1-4020-7444-1>
- [10] M. O’Neill and A. Brabazon, “Grammatical Swarm: The generation of programs by social programming,” *Natural Computing: an international journal*, vol. 5, no. 4, pp. 443–462, 2006.
- [11] P. C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger, “Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control,” *IEEE Symposium on Security and Privacy*, pp. 222–230, 2007.
- [12] D. E. Bell and L. J. LaPadula, “Computer Security Model: Unified Exposition and Multics Interpretation,” The MITRE Corporation, Bedford, MA. HQ Electronic Systems Division, Hanscom AFB, MA, Tech. Rep. ESD–TR–75–306, March 1976. [Online]. Available: <http://csrc.nist.gov/publications/history/bell76.pdf>
- [13] C. Ryan and R. M. A. Azad, “Sensible Initialisation in Chorus,” in *Genetic Programming, Proceedings of EuroGP’2003*, ser. LNCS, C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610. Essex: Springer-Verlag, 14-16 Apr. 2003, pp. 394–403. [Online]. Available: <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2610&page=394>
- [14] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.
- [15] Y. T. Lim, P. C. Cheng, J. A. Clark, and P. Rohatgi, “Policy Evolution with Genetic Programming: a Comparison of Three Approaches,” in *2008 IEEE Congress on Evolutionary Computation*, IEEE Computational Intelligence Society. Hong Kong: IEEE Press, 1-6 Jun. 2008, pp. 813–819.