MLS Security Policy Evolution with Genetic Programming

Yow Tzu Lim
Dept. of Computer Science
University of York
York, England, UK
yowtzu@cs.york.ac.uk

Pau Chen Cheng
Dept. of Security and Privacy
IBM Watson Research Center
Hawthorne, NY, USA
pau@us.ibm.com

Pankaj Rohatgi
Dept. of Security and Privacy
IBM Watson Research Center
Hawthorne, NY, USA
rohatgi@us.ibm.com

John Andrew Clark
Dept. of Computer Science
University of York
York, England, UK
jac@cs.york.ac.uk

ABSTRACT

In the early days a policy was a set of simple rules with a clear intuitive motivation that could be formalised to good effect. However the world is becoming much more complex. Subtle risk decisions may often need to be made and people are not always adept at expressing rationale for what they do. In this paper we investigate how policies can be inferred automatically using Genetic Programming (GP) from examples of decisions made. This allows us to discover a policy that may not formally have been documented, or else extract an underlying set of requirements by interpreting user decisions to posed "what if" scenarios. Three proof of concept experiments on MLS Bell-LaPadula, Budgetised MLS and Fuzzy MLS policies have been carried out. The results show this approach is promising.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning — Induction; D.4.6 [**Operating Systems**]: Security and Protection — Access controls; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems

General Terms

Experimentation

Keywords

Genetic Programming, Policy Inference, Security Policy, MLS

1. 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) [3]. 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.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

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 Genetic Programming (GP). It is a specific form of guided search that has seen success in a variety of domains.

This paper documents three proof of concept experiments inferring: MLS Bell-LaPadula, Budgetised MLS (a budgetised variant of MLS Bell-LaPadula) and Fuzzy MLS policies using GP. The results show this approach is promising.

The organisation of this paper is as follows: Section 2 documents the related work in this domain and Section 3 describes the general design of the experimental framework. Sections 4, 5 and 6 present the case study results for MLS Bell-LaPadula, Budgetised MLS and Fuzzy MLS policies respectively. Section 7 concludes the paper.

2. RELATED WORK

From the security perspective, security policies are mostly written in high level forms and later followed by a series of transformations and refinements. Improvement in easing the development include automated transformation of high level human understandable rules to low level machine executable rules, automated policy conflicts and coverage checking and resolution. There are no known attempts to generating the policy automatically from previous decision examples using machine learning techniques although autonomic security policies were mention in [6] (No results have yet appeared).

On the other hand, rule inferencing techniques have been around for many years in machine learning domain. There are various approaches proposed, e.g. decision tree induction, Genetic Algorithm (GA), Genetic Programming (GP), Artificial Immune Systems (AIS), etc. In this paper, emphasis is placed on GP. In [10] a grammar-based genetic programming system called LOGENPRO (The LOGic grammar based GENetic PROgramming system) is proposed. In the performance test conducted, it is found that LOGENPRO outperforms some Induction Logic Programming (ILP) systems. In [7] a GP experiment on co-evolution between rules and fuzzy membership variables is designed. The result shows that the output set of rules and variables are well adapted to one another. In [9] an attempt is made to invent a generic rule induction algorithm using grammar based GP. The result is shown to be competitive with well known manually designed rule induction algorithms. However, in all the above mentioned cases, there have been no previous known research on rule inferencing technique in the our domain of interest — security policy.

3. FRAMEWORK DESIGN

Most security policies can be represented as a set of `IF <condition> THEN <action>` rules. We shall attempt to discover an expression for the condition corresponding to some particular decision action (e.g. "allow read").

The leaf nodes at the bottom layer are elements of the terminal set T. At the next layer, leave nodes of the same type are joined together with operators that return *typed* values. These types may be numerical, set, vector or other user-defined types. At the next layer are the logic relational operators such as $<$ or $\in$; such an operator compares two typed values and returns a boolean value. These boolean values are combined at the higher levels with the logical composition operators such as AND, OR or NOT. The root node in a tree must evaluate to a Boolean. Strongly Typed Genetic Programming (STGP) [8] is used throughout the experiments presented in this paper. Figure 1 shows 3 examples of well-typed individual for the experiment in inferring MLS Bell-LaPadula policy for read access that will be presented in Section 4.

Using this representation, the security policy inference problem can be transformed into an N-class classification problem, in which N is the number of rules in the policy. STGP is used to search for the `<condition>` part of each rule. Therefore, the number of STGP runs increases linearly with the policy size. This is not as daunting as it seems to be. As all these searches are independent from one another, this design approach can benefit from the multicore processor revolution and execute searches in parallel. With only 1 processor, the binary decomposition method[1] can be employed to solve this problem in $N-1$ STGP runs.

The initial population are generated randomly using the *ramp half and half* method popularised by Koza [4]. This method takes the population size, minimum and maximum heights of the trees permitted in the population as inputs and generates approximately 50% trees with maximum height while the other 50% trees have heights between the minimum and the maximum heights.

The individual fitness is computed using decision examples in a training set. Each example is represented by a vector of variables which corresponds to the decision making factors and the decision itself. The fitness of the individual is based on the number of decision examples that it agrees with. In an ideal world, it might be desirable to match all examples. However, it is often the case in practice that there are a few poor decisions and many good decisions are made. The system might be expected to evolve a policy that agrees with the majority. 100% agreement is not essential. A lower degree of agreement may simply turn the spotlight on those specific individuals with decisions inconsistent with the inferred policy. In a sense, the fitness provides a measure of how well a candidate policy agrees with examined decisions, but also acts as anomaly detector. The accumulated total score after evaluating all examples becomes the individual's fitness score. Depending on the problem, all matches (or mis-matches) are not necessarily equal. The score given for a particular match or mis-match may have profound impact on the search process and the final result. Section 6 gives a good example.

After the fitness calculation stage, a new generation of individuals is produced with the use of evolutionary operators. Individuals with higher fitness scores have better chances to be selected to pass their "gene" (sub-tress) to the next generation. Further crossover and mutation operators are applied

[1]Binary decomposition method decomposes the N classes classification problem into $N-1$ binary classification problems. The first classification problem is $(c_1, {c_1}' \equiv P - c_1)$, second problem is $(c_2, {c_2}' \equiv {c_1}' - c_2)$, $N-1$ problem is $(c_{N-1}, {c_{N-1}}' \equiv {c_{N-2}}' - c_{N-1} \equiv c_N)$. The n^{th} binary classification problem can only be solved after the $n-1$ previous problems are solved. The algorithm is inherently sequential.

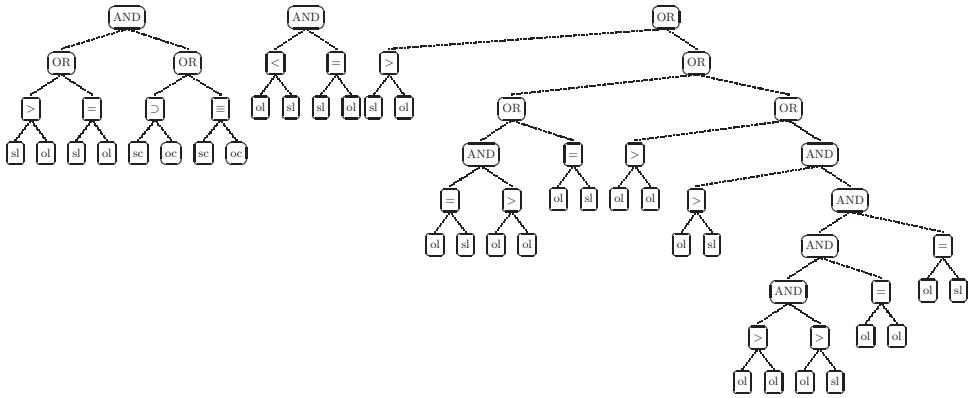

Figure 1: Examples of well-typed individuals. The leftmost individual resembles the condition for the read access in the MLS Bell-LaPadula policy, which is $(sl > ol\ OR\ sl = ol)\ AND\ (sc \supset oc\ OR\ sc \equiv oc)$. The other two are logically equivalent individuals. They resemble only the sensitivity aspect of this policy, which is $(sl > ol\ OR\ sl = ol)$

probabilistically. The evolution process continues until an individual with a "high enough" fitness score is found or a preset number of generations have elapsed.

4. MLS BELL-LAPADULA POLICY

In the first experiment, we concentrate on the "no read-up" part of the MLS Bell-LaPadula policy. It is simple, unambiguous and serves to demonstrate some interesting properties of our method of inference. For a read access, r, the policy can be summarised as:

$$\texttt{IF}\ sl \geq ol\ AND\ sc \supseteq oc\ \texttt{THEN}\ r\ \texttt{is}\ \textit{allowed} \quad (1)$$

$$\texttt{IF}\ sl < ol\ OR\ sc \not\supseteq oc\ \texttt{THEN}\ r\ \texttt{is}\ \textit{denied} \quad (2)$$

where sl and ol are subject and object sensitivity levels and sc and oc are subject and object category sets. Since the decisions are binary *allow/deny* decisions, the GP algorithm only needs to be run once to search for the condition for either *allow* or *deny*, the other condition can be simply obtained by logical negation. The condition for *allow* is chosen to be the learning target in this experiment.

The terminal set T consists of four variables, namely sl, ol, sc and oc but no constant value. The sl and ol are positive integers and tagged with the type "sensitivity", for which 3 operators are defined: $=$, $<$ and $>$. The $\leq$ and $\geq$ operators are intentionally omitted to make the search becomes more difficult. The sc and oc are sets and given the type "category", for which 3 operators are defined: $\equiv$, $\subset$ and $\supset$. The $\subseteq$ and $\supseteq$ operators are not included for the same reason given above. Each category in sc or oc is represented by a positive integer. The target condition, $TC(sl, ol, sc, oc)$ to be learnt in this experiment is:

$$(sl > ol\ OR\ sl = ol)\ AND\ (sc \supset oc\ OR\ sc \equiv oc) \quad (3)$$

In each run of the experiment, the maximum value of sensitivity levels for sl and ol, SNS_{max} and the total number of categories, CAT_{max} are defined. 100 randomly generated examples are used as the training set. Each example x is a vector with 5 attributes: sl_x, ol_x, sc_x, oc_x and dec_x. sl_x and ol_x are randomly chosen from $\{1\ ..\ SNS_{max}\}$; elements of sc_x and oc_x are randomly chosen from $\{1\ ..\ CAT_{max}\}$; and dec_x is set to be either 1 (*allow*) or 0 (*deny*) in accordance with the MLS Bell-LaPadula policy. Thus, all example decisions here are correct as far as MLS Bell-LaPadula policy is concerned.

The fitness of an individual (candidate policy), $fitness(i)$ is simply the sum of the matches between the decision made by the individual and the decision recorded in each example in the entire training set. Formally, let $d_{i,x}$ be the decision an individual i made for an example x; $True$ as 1; and $False$ as 0, then $fitness(i)$ is defined as in (4).

$$fitness(i) = \sum_{\forall\ example\ x} (d_{i,x} \equiv dec_x) \quad (4)$$

This experiment is carried out using the ECJ Framework v16 [5] and the experimental setup is summarised in Table 1. For those parameters that are not specified in Table 1, the default values defined in the framework are used.

Objective	Search for a TC logically equivalent condition in (3), which is $(sl > ol\ OR\ sl = ol)\ AND$ $(sc \supset oc\ OR\ sc \equiv oc)$
Terminal set T	$\{sl, ol, sc, oc\}$
Functional set F	$\{AND, OR, =, >, <, \equiv, \subset, \supset\}$
Fitness function $f(i)$	$\sum_{\forall\ example\ x} (d_{i,x} \equiv dec_x)$
Number of generations	50
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Mutation (0.1)
Maximum height of tree	17

Table 1: Experimental setup summary of MLS Bell-LaPadula policy inference for read access

Initially SNS_{max} and CAT_{max} are set to be 5. 10 runs of the experiment, each with a training set generated with a different random seed, are carried out. In all cases, logically equivalent conditions of TC in (3) can be learnt. Then we investigated the robustness of this inference technique as follows:

- scaling up SNS_{max} and CAT_{max};
- inclusion of "wrong" examples in the training set (i.e. inconsistent decision making is demonstrated);
- changing other parameters including population size and tree height.

4.1 Scaling Up SNS_{max} and CAT_{max}

The same experiment is repeated using 6 different settings of (SNS_{max}, CAT_{max}): $(10, 5)$, $(20, 5)$, $(30, 5)$ and $(5, 10)$, $(5, 20)$, $(5, 30)$. In the first 3 settings, in which SNS_{max} is scaled up to 30, TC can still be found. However, in the later 3 settings, in which CAT_{max} is scaled to 30, the result changes; only weaker conditions TC' are found. More precisely, the conditions learnt using the $(5, 10)$ setting are logically equivalent to $(sc \supset oc \; OR \; sc \equiv oc)$; the conditions learnt using the $(5, 20)$ setting are either logically equivalent to $(sc \supset oc \; OR \; sc \equiv oc)$ or simply $sc \supset oc$; and the conditions learnt using the $(5, 30)$ setting are logically equivalent to $sc \supset oc$.

Randomly generated categories sets pose interesting problems from a training point of view. The probability of randomly generating a pair (sc, oc) where $sc \equiv oc$ is small, $1/(2^{CAT_{max}})$ in fact. Thus the expected number of category equality examples in a sample of size N is $N/(2^{CAT_{max}})$. Unless the system sees examples of how equality should be handled, it cannot be expected to infer how that specific condition should be handled. Inference summarises rather than speculates. As usual, the training set characteristics are important. Experiments are carried out to validate this intuition. Examples are manually created to cover the equality case and the experiment is re-run with the same setting. The results agree with this intuition; logically equivalent conditions of TC are learnt for settings with CAT_{max} equals to 10, 20 and 30.

To further investigate the effect of training set coverage, 3 experiments with extreme settings are carried out. First, a training set with 9 examples that cover all the possible combinations of $((sl, ol), (sc, oc))$ relationships in TC, namely $(>, =, <) \times (\supset, \equiv, \subset)$ is used. The learnt condition is logically equivalent to TC. At the other extreme, an experimental setup using all examples with *deny* decision ($dec_x = 0$) yields the a logically equivalent condition of $False$. Conversely, if all examples in the training set are examples with *allow* decision, then a logically equivalent condition of $True$ is learnt. Thus, a mixture of correct *allow/deny* examples is required to evolve credible policies.

4.2 Inclusion of "Wrong" Examples

Here the experiment is repeated with the introduction of wrong examples in the training set (i.e. so we have examples of inconsistent decision making). We first started to introduce 10% "wrong" examples and then 20%, 25% and 30%. In all cases, the conditions learnt are similar with those learnt with all correct examples. This is because the search for the condition is guided by the fitness function which is defined as the number of matches between decisions made by an individual and the ones encoded in examples. In order to have maximum fitness, the search will tend to model the correct examples (which are in the majority) and choose to be inconsistent with the others. This is encouraging because 100% agreement is not the actual goal as mentioned earlier. Highlighting anomalous behaviours is also important.

4.3 Parameter Changes

Each experiment described so far is repeated using training set size consists of 500 and 1000 randomly generated examples. The conditions learnt in each experiment are very similar with the conditions learnt using only 100 examples.

Also, each experiment is repeated with various population sizes: 50, 100, 500 and 5000. When the population size is 500 or larger, logically equivalent conditions of TC are learnt, and there is no significant difference in terms of the number of generations required. However, the execution time per generation does increase significantly due to the increase in the amount of genetic operations performed. When the population size is set to be 50 and 100, the desired condition cannot be learnt sometimes. Investigation is made on these populations using the GUI provided by ECJ. Diversity in the population is lost in early generations, i.e. premature convergence in the population occurs.

To investigate the effect of tree size, the experiment is repeated with different maximum tree heights. In each experiment, the maximum tree height is set to be one less than that of the previous experiment; the first experiment has maximum tree height of 17. The target condition can not be learnt if the tree height is less than 4, which is the minimum height to represent TC. The results also show that the number of generations needed to learn the condition increases as the maximum tree height used increases, because larger tree height implies larger search space.

4.4 Read and Write Access

The experiment is extended to include write access in the MLS Bell-LaPadula model. For a write access w, MLS Bell-LaPadula policy can be summarised as:

$$\texttt{IF } sl \leq ol \;\; AND \;\; sc \subseteq oc \texttt{ THEN } w \texttt{ is } \textit{allowed} \quad (5)$$

$$\texttt{IF } sl > ol \;\; OR \;\; sc \not\subseteq oc \texttt{ THEN } w \texttt{ is } \textit{denied} \quad (6)$$

This is often known as the "no write-down" property (sometimes also known as the *-property).

We introduce a variable *access* denoting the type of requested access (*read* or *write*). The policy can be rewritten as follows:

$$\texttt{IF } (access = read \;\; AND \;\; sl \geq ol \;\; AND \;\; sc \supseteq oc) \;\; OR$$
$$(access = write \;\; AND \;\; sl \leq ol \;\; AND \;\; sc \subseteq oc)$$
$$\texttt{THEN } access \texttt{ is } \textit{allowed} \quad (7)$$

$$\texttt{IF } (access = read \;\; AND \;\; (sl < ol \;\; OR \;\; sc \not\supseteq oc)) \;\; OR$$
$$(access = write \;\; AND \;\; (sl > ol \;\; OR \;\; sc \not\subseteq oc))$$
$$\texttt{THEN } access \texttt{ is } \textit{denied} \quad (8)$$

This allows us to evolve the policy as a whole for read and write access.

The experimental setup is very similar as before with only a few minor changes. First, a new type, *access* is introduced to the type set and the terminal set is expanded to include a new variable, *access* and two new terminals, *read* and *write*; both of these have *access* as their types. The function set is extended to include $\leq$, $\geq$, $\subseteq$ and $\supseteq$ operators. Third, the training set size is increased to 500 randomly generated examples with the equality cases guaranteed as described in Section 4.1.

The target condition, $TC(access, sl, ol, sc, oc)$ to be learnt in this experiment is:

$$(access = read\ AND\ sl \geq ol\ AND\ sc \supseteq oc)\ OR$$
$$(access = write\ AND\ sl \leq ol\ AND\ sc \subseteq oc) \quad (9)$$

All other parameter settings including the fitness function remain the same as before. The experimental setup is summarised in Table 2.

Objective	Search for TC logically equivalent condition in (9), which is $(access = read\ AND\ sl \geq ol\ AND\ sc \supseteq oc)\ OR$ $(access = write\ AND\ sl \leq ol\ AND\ sc \subseteq oc)$
Terminal set T	$\{access, sl, ol, sc, oc\}$
Functional set F	$\{AND, OR, =, >, <, \equiv, \subset, \supset\} \cup \{\geq, \leq, \subseteq, \supseteq\}$
Fitness function $f(i)$	$\sum_{\forall\ example\ x} (d_{i,x} \equiv dec_x)$
Number of generations	50
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Mutation (0.1)
Maximum height of tree	17

Table 2: Experimental setup summary of MLS Bell-LaPadula policy inference for read and write access

Initially SNS_{max} and CAT_{max} are set to 5. 10 runs of the experiment, each with a training set generated with a different random seed, are carried out. In all cases, logically equivalent of TC in (9) can be learnt. We now investigate the robustness of this inference technique as before.

4.5 Scaling Up SNS_{max} and CAT_{max}

The SNS_{max} and CAT_{max} scaled up in the similar fashion as in previous experiment using 6 different settings of (SNS_{max}, CAT_{max}): $(10, 5)$, $(20, 5)$, $(30, 5)$ and $(5, 10)$, $(5, 20)$, $(5, 30)$. As the training set used covers the category equality cases, the logical equivalent of TC in (9) is learnt in all cases.

4.6 Inclusion of "Wrong" Examples

As in previous experiments, we introduced 10%, 20%, 25% and 30% "wrong" examples. In all cases, the conditions learnt are similar with those learnt with all correct examples. With 30% wrong examples, the number of successful runs decreases to 7 out of the 10 runs. Investigation of the best individuals in unsuccessful runs shows that these individuals generally represent a slightly weaker or stronger conditions, $TC\prime$. Some of these individuals after boolean simplification are shown as follows:

$$(access = read\ AND\ sl \geq ol\ AND\ sc \supseteq oc)\ OR$$
$$(access = write\ AND\ sl \leq ol\ AND\ sc \subseteq oc)\ OR$$
$$(sc = oc) \quad (10)$$

$$(access = read\ AND\ sl > ol\ AND\ sc \supset oc)\ OR$$
$$(access = write\ AND\ sl < ol\ AND\ sc \supseteq oc) \quad (11)$$

Having said that, the number of examples these individuals agree with is more than 450 (90% and above) examples.

4.7 Parameter Changes

Each experiment described so far is repeated using training set size consists of 1000 randomly generated examples. In all cases, the logically equivalent conditions of TC are learnt.

Also, this experiment is repeated with population sizes of 50, 100, 500 and 5000. When the population size is 500 or larger, logical equivalent of TC can be learnt, and there is no significant difference in terms of the number of generations required. However, the execution time per generation does increase significantly due to the increase in the amount of genetic operations performed. When the population size is set to be 50 and 100, the desired condition cannot be learnt sometimes. Investigation is made on these populations and using the GUI provided by ECJ. Diversity in the population is lost in the early generations, i.e. premature convergence in the population occurs.

In investigating the effect of tree size, it is found that TC can not be learnt if the tree height is less than 5. This is obvious as this is minimum tree height necessary to represent TC. As the tree height increases, the number of generations needed to learn the condition increases because larger tree height implies larger search space.

5. BUDGETISED MLS POLICY

In this experiment, we designed a new risk-based policy with intuition drawn from [1, 3]. For a read access r, Budgetised MLS policy is as follows:

$$\texttt{IF}\ pos(ol - sl)\ +\ \#(oc \setminus sc)) \leq budget$$
$$\texttt{THEN}\ r\ \texttt{is}\ \textit{allowed} \quad (12)$$
$$\texttt{IF}\ pos(ol - sl) + \#(oc \setminus sc)) > budget$$
$$\texttt{THEN}\ r\ \texttt{is}\ \textit{denied} \quad (13)$$

where $pos(x)$ returns x if $x >= 0$ or 0 if $x < 0$; $x \setminus y$ is the set difference between set x and set y; $\#(x)$ is the cardinality of the set x and *budget* is the amount the requester is willing to pay for the requested access. The target condition TC for this experiment becomes

$$pos(ol - sl) + \#(oc \setminus sc) \leq budget \quad (14)$$

The experimental setup is similar to the previous experiment except with the addition of 8 operators (pos, $\#$, $\setminus$, $+$, $-$, $*$, $/$, exp) and is summarised in Table 3.

The results show that TC can be learnt in all cases. The investigation on the robustness of this inference technique by scaling up SNS_{max} and CAT_{max}, inclusion of "wrong" examples in the training set, changing other parameters produces similar results as in previous experiments. The only difference is the minimum tree size required now is increased to 8 due to the complexity.

6. FUZZY MLS POLICY

Up to this point, we have only considered binary decision making. In this experiment, we concentrate on learning the conditions of a risk-based policy that could have more than two decisions, beyond the *allow* and *deny* binary decision model. The policy model is the Fuzzy MLS model [1, 2].

Objective	Search for a logically equivalent condition of TC in (14), which is $pos(ol-sl)+\#(oc \setminus sc) \leq budget$
Terminal set T	$\{sl, ol, sc, oc, budget\}$
Functional set F	$\{AND, OR, =, >, <, \equiv, \subset, \supset\} \cup \{\geq, \leq, \subseteq, \supseteq\} \cup \{pos, \#, \setminus +, -, *, /, exp\}$
Fitness function $f(i)$	$\sum_{\forall\ example\ x} (d_{i,x} \equiv dec_x)$
Number of generations	50
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Mutation (0.1)
Maximum height of tree	17

Table 3: Experimental setup summary of Budgetised MLS policy inference for read access

This policy uses the risk-based rationale of the MLS Bell-LaPadula policy to compute a *quantified risk estimate* by quantifying the "gap" between a subject's label and an object's label in an MLS system.

Quantified risk estimates are numbers and therefore could be used to build the risk scale shown in Figure 2 [1]. The risk

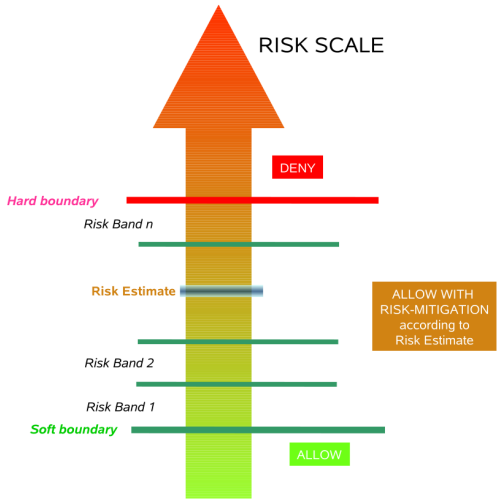

Figure 2: Risk adaptive access control on a risk scale

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 different certain risk mitigation measures*.

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

$$risk = (value\ of\ damage) \times (probability\ of\ unauthorised\ disclosure) \quad (15)$$

The value of 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 same[2], and thus risk becomes a function of subject's and object's sensitivity levels only. For more detail on risk quantification, refer [1, 2].

Since there is no discussion on the way to partition the scale into risk bands in [1], the following formula is defined to map a risk number to a risk band:

$$band(risk(sl, ol)) = min(\lfloor log_{10}(risk(sl, ol)) \rfloor, N) \quad (16)$$

where $(N+1)$ is the number of bands desired on the scale and the function $risk(sl, ol)$ is defined in Appendix A according to [1]. Base-10 logarithm is used in (16) to compute the order of magnitude of risk as the band number. In our experiments, the scale is divided into 10 bands ($N = 9$) numbered from 0 to 9. Therefore, 10 STGP runs are required to search for conditions for all the bands. This experiment serves as a good illustration of the N-classes classification problem discussed in Section 3 such that each band corresponds to a class.

Since only the sensitivity levels are considered, the terminal set T has only two variables, namely sl and ol. However, an additional set of real constant number in the range of $(-1, 1)$ is added to T to satisfy the sufficiency property. This is because the target condition, TC can no longer be represented with only sl and ol. Elements in T are given type *sensitivity*, for which 8 operators are defined: 3 relational operators ($=$, $<$ and $>$) which returns a value of type "boolean" and 5 arithmetic operators $(+, -, *, /, exp)$ which returns a value of type "sensitivity". The target condition for band j, TC_j to be learnt is:

$$TC_j(sl, ol) = (band(risk(sl, ol)) \equiv j) \quad (17)$$

In other words, STGP is used to search for an equivalent function of the composition of 2 functions: the band function and risk function.

For all experiments described in this section, SNS_{max} is set to be 10 and an example x in a training set is a triple $(sl_x, ol_x, band_x)$, where $band_x = band(risk(sl_x, ol_x))$. In the first experiment, we generated a training set with $10 \times 10 = 100$ examples that cover all the possible (sl, ol) pairs and see what can be learnt in this optimal setting. The numbers of examples in each band is shown in Table 4. Band 0 has the most number of examples because it includes all the low-risk cases where $sl > ol$. All the other bands have relatively small number of examples.

As in the MLS Bell-LaPadula Experiment in Section 4, the fitness score for an individual in the search for condition of band j, $fitness_j(i)$ is defined to be the total number of correct decisions i made:

$$fitness_j(i) = \sum_{\{x|band_x \equiv j\}} (d_{i,x} \equiv True) + \sum_{\{x|band_x \not\equiv j\}} (d_{i,x} \equiv False) \quad (18)$$

The experimental setup is summarised in Table 5. As before, this experiment is carried out using the ECJ Frame-

[2]Therefore the gap between categories sets is 0.

Band	Number of examples
0	52
1	5
2	3
3	4
4.	3
5	5
6	5
7	7
8	8
9	8

Table 4: Distribution of examples in the training set

work v16 [5] and the default values are used for the unmentioned parameters.

Objective	Search for condition that classifies all examples into a risk band j accurately, which is, $\forall x$, $band(risk(sl_x, ol_x)) \equiv band_x$
Terminal set T	$\{sl, ol\} \cup \{r \mid -1.0 < r < 1.0\}$
Function set F	$\{AND, OR, =, >, <\} \cup \{+, -, *, /, exp\}$
Fitness function $f_j(i)$	$\sum_{\{x \mid band_x \equiv j\}}(d_{i,x} \equiv True) + \sum_{\{x \mid band_x \not\equiv j\}}(d_{i,x} \equiv False)$
Number of generations	500
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Reproduction (0.1)
Maximum height of tree	17

Table 5: Experimental setup summary of Fuzzy MLS policy inference for read access

For each band, 10 runs are conducted. However, the result shows that only the target condition for band 0, TC_0 can be learnt.

Investigations on the experimental logs for other bands reveals the following facts:

- the fittest individuals remain unchanged after a few generations.
- individual fitness scores converge quickly to that of the fittest individual. This suggests that the searches for band in {1 .. 9} quickly become random searches because equivalent fitness implies equivalent probability one individual to be selected.
- for a band in {1 .. 9}, the fitness score of the fittest individual is $100-(number\ of\ examples\ in\ the\ band)$. This suggests that all points in fitness score come from negative examples (example not in that band in question), implying these examples completely outweigh the relative few examples in that band. This also explains why TC_0 can be learnt since more than half of the examples are in band 0.

6.1 Weight and Punishment on Fitness Score

To improve the result, we assign different weights to the fitness scores of different kinds of decisions made by an individual according the following principles. In the search for condition of band j, TC_j:

- For a correct decision, *award more* if
  - the risk is higher for security concerns; i.e., award more for a larger j.
  - the decision is *is true positive* (hits the target); i.e., award more when $band_x \equiv j$. This will overcome the effect that relative few positive examples are in band j when $j \neq 0$.
- For an incorrect decision, *punish more* if
  - the decision is *false positive* ($d_{i,x} \equiv True$ and $band_x \neq j$) and is *more off the target*; i.e., punish more as $\mid j - band_x \mid$ becomes larger. Also, for security concerns, punish more if this false positive decision *underestimates the risk*; i.e., punish more if $band_x > j$.
  - the decision is *false negative* ($d_{i,x} \equiv False$ and $band_x = j$) when the risk is higher for security concerns; i.e., punish more for a larger j.

Using these principles, the fitness function is changed to be:

$$\begin{aligned} fitness_j(i) \quad = \quad & \sum_{\{x \mid band_x \equiv j\}} w_{tp}\{d_{i,x} \equiv True\} \; + \\ & \sum_{\{x \mid band_x \not\equiv j\}} w_{tn}\{d_{i,x} \equiv False\} \; - \\ & \sum_{\{x \mid band_x \not\equiv j\}} w_{fp}\{d_{i,x} \equiv True\} \; - \\ & \sum_{\{x \mid band_x \equiv j\}} w_{fn}\{d_{i,x} \equiv False\} \end{aligned} \tag{19}$$

where

$$\begin{aligned} w_{tp} &= j+1, \\ w_{tn} &= (j+1)/10, \\ w_{fp} &= \begin{cases} band_x - j & \text{if } band_x > j, \\ (j - band_x)/2 & \text{if } band_x < j, \end{cases} \\ w_{fn} &= j+1 \end{aligned}$$

The experiment is re-run using the new fitness function on the same training set. The result shows that the weights and punishments are indeed very useful and the correct conditions can be learnt for all 10 bands. As before, the effects of the population size and the tree height are investigated.

6.2 Parameter Changes

This experiment is repeated with different population sizes of 50, 100, 500 and 5000. Logically equivalent conditions of TC_j for all 10 bands can be learnt if the population size is set to be 500 or greater. When the population size is small (50 or 100), the target conditions (TC_j) of some bands cannot be learnt sometimes. Investigation made on the individuals gives the similar result with the MLS Bell-LaPadula experiment — the phenomena of premature convergence in the population occurs.

As before, this experiment is re-run with different value for maximum heights, ranging from 1 to 17. The target conditions (TC_j) for all 10 bands cannot be learnt when the maximum tree height becomes less than 10. This is significantly larger than that in the MLS Bell-LaPadula experiment due to the complexity of the function it tries to model.

7. CONCLUSION AND FUTURE WORK

Security policy inference is a new domain in which evolutionary algorithms can be employed. In this paper, we proposed a generic policy inference framework using the classification approach, implemented using Genetic Programming. Three experiments on MLS Bell-LaPadula, Budgetised MLS and Fuzzy MLS policies are conducted to validate the proposal. The results show that this approach is promising and the correct policy can be inferred from examples.

8. ACKNOWLEDGEMENTS

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.

We would also like to thank Aaron Kershenbaum, Dakshi Agrawal of IBM Watson T. J. Watson Research Center, Hawthorne in providing valuable feedback and facilities to conduct this research.

9. REFERENCES

[1] 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. Technical report, IBM Research Report RC24190, 2007.

[2] 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*, pages 222–230, 2007.

[3] Horizontal Integration: Broader Access Models for Realizing Information Dominance. Technical Report JSR-04-132, The MITRE Corporation JASON Program Office, Mclean, Virginia, Dec 2004.

[4] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[5] S. Luke. ECJ version 16 A Java-based Evolutionary Computation Research System, August 2007.

[6] P. D. McDaniel. Policy Evolution: Autonomic Environmental Security, December 2004.

[7] R. R. F. Mendes, F. de B. Voznika, J. C. Nievola, and A. A. Freitas. Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 183, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[8] D. J. Montana. Strongly Typed Genetic Programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[9] G. Pappa and A. Freitas. Towards a genetic programming algorithm for automatically evolving rule induction algorithms. In J. Furnkranz, editor, *Proc. ECML/PKDD-2004 Workshop on Advances in Inductive Learning*, pages 93–108, Pisa, Italy, September 2004.

[10] M. L. Wong and K. S. Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Kluwer Academic Publishers, Jan. 2000.

APPENDIX

A. $RISK(SL, OL)$

In [1], 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:

$$risk(sl, ol) \;=\; Val(ol) \times P_1(sl, ol) \tag{20}$$

$Val(ol)$ is the estimate value of damage and is define in [1] as

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

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 [1] 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 [1]. 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 access to an object with sensitive level equals to or more than M should be controlled by human beings and not machines. In our experiments, the maximum value (SNS_{max}) for sl and ol is 10; the settings for k, mid and M are $k = 3,\ mid = 4,\ M = 11$.