

Learning Autonomic Security Reconfiguration Policies

Juan E. Tapiador and John A. Clark

Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK
{jet, jac}@cs.york.ac.uk

Abstract—We explore the idea of applying machine learning techniques to automatically infer risk-adaptive policies to reconfigure a network security architecture when the context in which it operates changes. To illustrate our approach, we consider the case of a MANET where nodes carrying sensitive services (e.g., web servers, key repositories, etc.) should consider relocating themselves into a different node to guarantee proper functioning. We use simulation to derive properties from a candidate policy, and then apply Genetic Programming and Multi-Objective Optimisation techniques to search for optimal candidates. The inferred policies take the form of risk-aware service relocation algorithms that autonomously dictate when and how to relocate services with the aim of keeping risk to a minimum. Since security policies often have implications in dimensions other than security, we force the learning process to consider also the consequences (performance, usability) of a given policy.

Index Terms—Security Policy Inference; Risk Management; Mobile Ad Hoc Networks; Genetic Programming.

I. INTRODUCTION

Formulating a good security policy is an intrinsically difficult problem. Traditional approaches rely on the experience and intuitions of security administrators who, equipped with an appropriate knowledge of the system, the foreseeable risks, and the potential (and affordable) countermeasures, produce a policy making reasonable tradeoffs between security and other relevant criteria. It has been argued that neither such a methodology nor the resulting policies could be appropriate for certain networking paradigms (e.g., MANETs) where multiple autonomous systems interact with each other in unpredictable ways. A notable example is found in military operations that require the formation of dynamic mobile networks without relying on preexisting infrastructures. Many of the constituent elements of such communication systems (for example, the basic units providing the physical connection layer upon which data is transmitted) frequently need to be moved about a geographical area as the mission evolves. Unlike most civilian applications, the majority of military networks operate against a complex, hostile background wherein the security of the

nodes and the information (both stored and transmitted) is paramount. At a high level, protection is provided through a security architecture which defines aspects such as what services are located in what nodes; what subnetworks exist and how they are interconnected; what security controls must an information flow pass through before reaching a node; etc. Traditional approaches to design and deploy security architectures are predominantly static, done by a human security administrator, and often based on risk analysis methodologies that rely on stationary assumptions about the network structure, typical information flows, potential threats posed by the adversaries, and location and configuration of security services.

One readily sees that few or none of these assumptions hold in a MANET setting. If the conditions under which a network operates change and, consequently, the current security configuration is no longer optimal (or, simply, not good enough), one desirable response would be for the network to reconfigure itself and adapt to the new situation. This notion of *adaptation* has been a central theme in autonomic computing almost since its inception. When it comes to security properties, one possible way of implementing adaptive systems is through a security policy that takes into account the context in which the system operates. Implicit in such an approach is the assumption that one can foresee all possible situations and, therefore, derive rules dictating how to best respond. This is clearly unrealistic and other approaches must be explored. In this work, we present an approach based on an entirely different approach: security policies can be *learned* rather than specified.

A. Motivation and Overview

As proof of concept, in this work we will consider the case of a tactical MANET whose nodes are moving about a physical terrain. Parts of the terrain will be considered more susceptible to eavesdropping or physical attack than others. There will be one (or more) indicators of risk associated with being at a particular point in the terrain. Thus, as nodes move about risk exposure changes.

We propose to use guided search and machine learning approaches to discover risk-aware service relocation algorithms. Such algorithms will dictate when a service needs to be relocated and how. In particular, we will use Genetic Programming (GP) to search for a program that each node carrying sensitive services will execute. The program will gather some information about the current networking and physical context and will decide whether it is safe to maintain

This research was sponsored by the U.S. Army Research Laboratory and the U.K. 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 U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

the service in the current node or else it would be convenient to migrate to a different one. Roughly speaking, our approach is based on the idea of finding a valuation function that will measure the suitability of a given node to provide a service. Implicit in such a function should be the entire security policy, so that nodes under dangerous conditions should receive a low value and vice versa. This would suffice when simple risk reduction is the sole goal. However, we will also have to extend the scheme to capture other properties of interest, particularly those related to network performance (e.g., load balancing). Otherwise, the resulting security policy might result in totally unusable network configurations.

The rest of this work is organised as follows. In Section II we discuss some related works where GP and other forms of heuristic search have been used to derive security components. In Section III we provide a brief background on GP systems. Section IV describes a scenario that will be used throughout this paper to illustrate our ideas, and also the simulation environment used for experimentation. In Section V we introduce the concept of reconfiguration policy for MANET services and describe its main components, including those which must be learnt. In Section VI we describe the experiments carried out and discuss some policies found using different goals as evaluation criteria. Finally, Section VII concludes the paper.

II. RELATED WORK

Heuristic search and other techniques taken from Artificial Intelligence have been successfully applied to various problems related to the design and analysis of access control policies. For example, a central problem in role-based access control is to identify the set of roles and their associated privileges. Various authors (see e.g. [10], [12], [22]) have explored how data mining can be of help in this process, automatically engineering roles from existing data. In a related series of works [13], [14], [15], Lim et al. show how GP (and subsequently Grammatical Evolution) can be used to infer an access control policy from a set of decision examples. The authors explore the inference of generic Multi-Level Security (MLS) policies coupled with the Fuzzy MLS risk model proposed in [2]. It is shown how GP has the ability to correctly capture the implicit policy provided by the examples. In [15], the approach is attempted in an online training setting, where the access control policy changes over time and the system must be continuously learning.

The inference of security policies using automatic means has been explored in other domains too. The idea of *evolving* a security policy for an autonomic system which must consider environmental factors was briefly mentioned in the work of McDaniel [19], yet no results were published. Tongaonkar et al. [25] have recently shown how high-level network access policies can be derived from low-level firewall filtering rules. Similarly, Scott et al. [23] discuss how to automatically infer the security policy of a web gateway by observing the interaction between the web server and its clients. Inoue et al. describe in [4] how to learn application-specific Java sandboxing policies by monitoring the program execution and

learning from it.

A common characteristic of all these works is that they just focus on learning a policy either from examples provided by a human operator or from data observed during the real-time operation of the system. But a security policy often causes important effects on non-security parameters of the system, notably on its usability and performance. Whilst it is widely recognised that some tradeoffs inevitably exist, characterising them is an entirely different matter. We believe that such consequences should also be taken into account in the inference process in order to obtain not only a good security policy, but a *usable* one.

III. GENETIC PROGRAMMING

Evolutionary algorithms are search techniques inspired by various evolutionary processes and principles [7], [3]. The search generally starts with an initial population of individuals, each one representing a potential solution to the problem at hand. This initial population is generated either randomly or by any other means. Associated with each individual is a “fitness” or “cost” value that measures how well this candidate solution solves the problem. Once every individual is evaluated, the population is repeatedly subjected to a series of “evolutionary operators:”

- Selection: a subset of individuals is selected for breeding according to their fitness: the higher the fitness, the higher the chance of being selected.
- Crossover: selected individuals are used to generate new individuals resembling the “parents.”
- Mutation: elements within a solution are somehow perturbed. This introduces diversification into the population, allowing to explore parts of the search space which would be unreachable by selection and crossover alone.
- Reproduction: some exceptional individuals survive to the next generation without changes.

The repeated application of this process generates successive populations composed of solutions which are increasingly better. The search stops either when a good enough solution is found or after a predefined number of generations.

GP is an evolutionary search technique where individuals in the population are programs [9], [11]. Such programs are often represented using tree structures, where the terminal nodes are constants and variables, and the internal nodes are functions, operators, statements or control structures of the programming language. When the program is a function returning a value, the root of the tree represents the final value computed. For example, Fig. 1 shows a simple tree representing the expression $F(X, Y, Z) = X * \frac{Y}{13} + \min(0.5, \sin(Z))$. A program is evaluated by recursively parsing the tree, so each node is evaluated and the value is passed to the parent until the entire program is executed. Note that more complex control structures such as if-then statements or loops can be easily represented too using trees or other data structures. In GP, crossover is often implemented over two trees: a sub-tree in each parent is randomly chosen and swapped with the other, so two new trees are generated. Mutation is generally performed

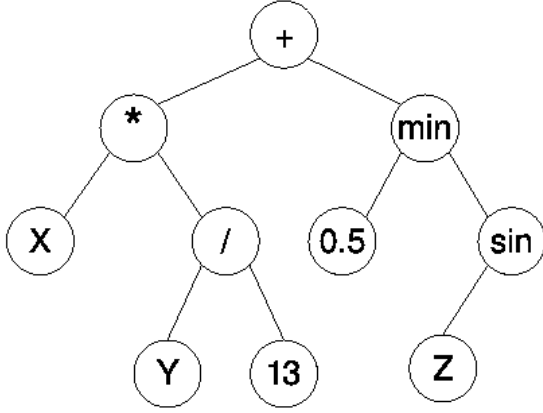


Fig. 1. Example of a tree structure representing the function $F(X, Y, Z) = X * \frac{Y}{13} + \min(0.5, \sin(Z))$.

on one tree by randomly choosing a node and replacing it with a new subtree which is also generated randomly.

IV. SCENARIO

We next describe the scenario and simulation environment used in our experiments.

A. Network

We will consider a MANET composed of N nodes moving about a physical terrain. Each node acts either as a server or as a client. Servers provide specific network services, such as e-mail, key repositories, HTTP or FTP services, etc. For each service, there will be one or more identical servers distributed across the network, so when a client needs to access a service it does it through the closest server. In this way, for each service the network is divided into a number of clusters, each one being served by a server. This is a common and scalable approach in many settings where load balancing is necessary, or where large blocks of the network can be temporarily disconnected from each other due to mobility and physical constraints.

B. Geographical Risk

The risk measure that we will consider is geographical in nature and motivated by the operation of a MANET in hostile territory. Parts of the terrain might be under control of the adversary, so a node approaching them would see increased its likelihood of being captured, eavesdropped, jammed, physically attacked, etc. We will not discriminate between different threats, and will assume that a probability indicator, $P_{attack}(x, y, z) \in [0, 1]$ can be obtained by a node *only if* the node is at location (x, y, z) . This assumption holds in many scenarios where enough intelligence about the terrain and the adversary cannot be gained in advance, and also in missions where the trajectories of the nodes and the location of the adversaries change over time unpredictably.

The value of P_{attack} can be computed in different ways depending on the specific attack(s) we are interested in

avoiding. For example, appropriate detection measures provide valuable real-time information about the attacker's actions (e.g., attack attempts detected by an IDS, flows blocked by a firewall, jamming signals detected by a listening device, etc.) This feedback could be easily summarised into a probability measure. Other forms of risks can be also considered (see for example [1] for an excellent survey of various risk measures relevant in a MANET context).

Note that we have explicitly made the geographical risk independent on the node. Consequently, we will interpret it as the *probability* of something undesirable happening, regardless of the value or function of the node. In order to obtain a true risk measure, such a probability has to be multiplied by the value of the service and/or the node. We will denote by

$$Risk(n, x, y, z) = Val(n) \cdot P_{attack}(x, y, z) \quad (1)$$

the risk of node n at position (x, y, z) . The value $Val(n)$ of node n is a positive quantity proportional to its importance (or, equivalently, to the loss we incur in by losing it).

C. Configuration Performance: Load Balancing

As mentioned before, one reason for deploying various identical copies of the same server is to balance the network load and to ensure service availability when the network suffers fragmentation. In order for such a strategy to be effective, the replicas must be conveniently distributed across the network. Next we introduce an indicator to measure this property.

For clarity, assume that only one network service is provided by a set $\mathcal{S} = \{s_1, \dots, s_{|\mathcal{S}|}\}$ of servers. At a given time instant, let $\mathcal{C}(s_i)$ be the set of clients associated with server s_i ; that is, those nodes for which s_i is the closest server according to the routing metric. If $N - |\mathcal{S}|$ is the total number of clients in the network, then in an ideal configuration each server should serve exactly $\frac{N - |\mathcal{S}|}{|\mathcal{S}|}$ clients (rounding up or down appropriately). Consequently, the quantity

$$B = \frac{1}{N} \sum_{i=1}^{|\mathcal{S}|} \left| |\mathcal{C}(s_i)| - \frac{N - |\mathcal{S}|}{|\mathcal{S}|} \right| \quad (2)$$

is minimised when the placement of the servers is optimal. (Note that the normalisation constant ensures that $B \in [0, 1]$.)

We will use B as an indicator of the quality of the current configuration: the better the configuration, the lower the value of B . Note that we are implicitly assuming that the probability of a client accessing a server is uniform among all the clients; otherwise the communication patterns should be taken into account to produce a weighted measure. Likewise, an optimal configuration does not guarantee full availability against fragmentation, for that depends on factors that we assume unpredictable (e.g., mobility patterns, radio propagation barriers caused by the landscape, etc.).

D. Simulation Environment

Here we describe a concrete scenario and the simulation environment that will be used later in the experimentation. We

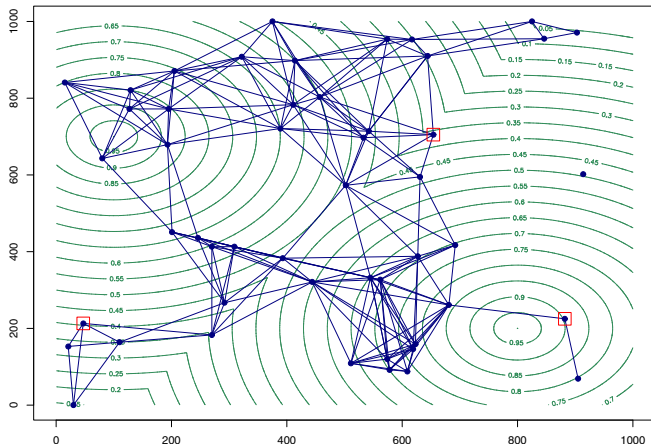


Fig. 2. (In colour in the electronic version.) Snapshot of a simulation of a MANET moving about a hostile terrain.

will consider a MANET composed of 50 nodes moving about a $1000\text{m} \times 1000\text{m}$ area. The network contains 3 servers which offer the same service, so the remaining 47 nodes connect to the closest server each time an access is required. The scenario is implemented in the NS-2 network simulator [20] and we use AODV [21] as routing protocol. Initial positions and movement patterns are randomly produced using the CMU scenario generator included within NS-2, with maximum speeds varying between 5 and 15 m/s, and pause time set to 0 (i.e., nodes are constantly moving). The total simulation time is 200 s.

In absence of real-life (and possibly mission-specific) risk data, we have used an artificial construction to represent a hypothetical risk map. This consists of two highly risky spots located at coordinates (100, 700) and (800, 200) in our 1000×1000 grid. Each point¹ (x, y) is then assigned a probability-of-attack value which depends on the distance to these points: the farther away from them, the lower the probability. We will assume that all the servers have the same value equal to 1 unit, so the risk will be simply the probability of attack. This risk map is constant, i.e., it does not change over time. Fig. 2 shows a snapshot of a simulation of this scenario. Each blue point represents a node and lines illustrate the current connectivity status. Servers are enclosed within a red square. The background surface (in green) represents the risk level of the terrain, with the two adversary locations easily discernible.

V. RECONFIGURATION POLICIES

The relocation of services in a MANET obeys a simple principle: a server under too much risk should consider migrating to a different node with lower risk. A reconfiguration

¹For simplicity, we have used a 2D terrain, so no z coordinate will be specified.

policy should establish *what*, *when* and *where* to move. We make three basic assumptions concerning this process:

- 1) Transferring the server (program) and all additional data required for proper functioning to other node is possible but very costly, so the number of relocations should be kept down.
- 2) Risk is not the only property to be managed. Performance should not be neglected.
- 3) All nodes are equally valid to accommodate a server and collaborate towards this end: a node chosen to house a server cannot refuse to do it.

Our approach is based on a relocation protocol run by each server. Let $\mathcal{C}(s) = \{c_1, \dots, c_k\}$ be the set of nodes which can house server s when it needs to be moved².

One basic task for the server is to decide which of them is the best. To make this decision, the server periodically collects from each c_i a number of variables, or indicators, m_1^i, \dots, m_d^i . In our scheme, the server broadcasts a request and every c_i sends back its indicators. In order to keep the protocol efficient, we will assume that these indicators are local to each node; that is, each c_i can produce them without exchanging information with other nodes. Some examples may be the local risk of each node, their position within the network, the distance to the closest server, the number of neighbours; the node's computing power and remaining battery, etc. Once these indicators are collected, the server computes for each candidate node a function $f_i = f(m_1^i, \dots, m_d^i)$, and the result is interpreted as how suitable c_i is to house the server: the higher f_i , the more suitable c_i is. This solves the question of *where* to move, as the server will move to the most suitable node.

The remaining important decision is *when* to move. One possible solution is to devise an additional function (or set of rules) to make such a decision. This, however, is not necessary at all, as the temporal dimension could be implicitly incorporated into the suitability function f : the server will move as soon as a more suitable node is found. More formally, s will move to node $j = \arg \max_i \{f_i\}$ iff $f_j > f_s$. Note that this approach does not necessarily imply a large number of relocations, since that will depend on the particular form of f and the environment where the nodes operate.

A. Learning through Simulation

Assume that we are given a simulator that allows us to plug in a candidate policy f and study what effects it has on a network over a given period of time. We next discuss how such an f can be evaluated in terms of the outcomes of the simulation. The ability to do this will allow us to compare different policies and will be instrumental in learning optimal policies.

During the simulation of f , we will be interested in three main outcomes. The first is a measure of how f works in

²We assume that the nodes which can house a server are only the clients associated with it at a given time, hence the use of the same notation that in Section IV-C. This ensures that the reconfiguration is feasible, as the network could be fragmented and others nodes might not be reachable.

terms of risk. At each time step, we will record the position of each server and will compute their current risk according to expression (1). At the end of the simulation, the average $R(f)$ of all the individual measures will be used as the risk measure of f .

In order to study performance, we will use the balance indicator B given by (2). We will denote by $B(f)$ the average of the balance values computed over the entire simulation period.

Finally, relocation is a mechanism that should not be used too often, so we will also keep track of how many relocations a policy generates. We will denote by $L(f)$ the average number of relocations per time unit.

Each of these three values $R(f)$, $B(f)$, and $L(f)$ constitute independent objectives that should be minimised by a good policy. One simple way of summarising them into a single measure consists of using a weighted sum of the form

$$G(f) = w_1 R(f) + w_2 B(f) + w_3 L(f) \quad (3)$$

where each $w_i \geq 0$ weights the relative importance placed on each goal. $G(f)$ can be thus viewed as a cost measure: the lower the value of $G(f)$, the better the reconfiguration policy induced by f .

VI. EXPERIMENTS

We next describe and discuss some experimental results obtained for the scenario described in Section IV-D.

A. Experimental Design

Our goal is to search for an optimal (lowest possible cost) suitability function f . Each individual in the population encodes a candidate function. The terminal set consists of Ephemeral Random Constants (ERC) –constant whose value is randomly generated during their creation– in the interval $[-1, 1]$, and three additional variables that each server must collect from each candidate destination:

- r : node’s local risk.
- g : node’s degree (number of neighbours).
- d : node’s distance to its closest server.

We experimentally determined that these three indicators are the most adequate inputs to the policy. (In fact, they constitute reasonable factors upon which decide when and where to move.)

Policies (i.e., suitability functions) will be represented as Lisp-like S-expressions. (See e.g. [24] for an early use of S-expressions in SDSI and SPKI). The function set is composed by usual arithmetic operations (+, -, *), trigonometric (sin, cos) and exponential (exp) functions, and protected³ versions of division and logarithm. Limits are imposed on the size of the programs: the maximum allowed depth of the tree is 13 and the maximum number of nodes is 150. This design allows us to represent a reasonably vast set of candidate functions. Note that we do not enforce the function to satisfy

$$^3x/y = \begin{cases} \frac{x}{y} & \text{if } y \neq 0 \\ 1 & \text{otherwise} \end{cases} \quad \text{and} \quad \log(x) = \begin{cases} \log(|x|) & \text{if } x \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

any constraints: the evolutionary process will hopefully discard those candidates that for whatever reason do not behave well.

To evaluate the cost of a function f , the simulation is executed using f at the core of the reconfiguration policy as explained before. During the simulation, the relevant statistics are harvested (see Section V) and returned when the simulation finishes. A cost value $G(f)$ is then computed using expression (3). In order to increase the robustness of the learning process, the policy given by f is executed 10 times in the same scenario with different random initialisations and movement patterns. The final cost value associated with f is then the average of the 10 cost values obtained in each single simulation.

We follow the common practice of reformulating the minimisation problem into a maximisation one by using an adjusted fitness function of the form $A(f) = \frac{1}{1+G(f)}$, where $G(f)$ is the original cost function, so now the goal is to maximise $A(f)$. This formulation presents the advantage that the fitness is now normalised between 0 and 1, and also small improvements in $G(f)$ are amplified in $A(f)$.

Other relevant parameters of the search are: the maximum number of generations is set to 100; the population size is 200 tress; and crossover and mutation probabilities are 0.9 and 0.1, respectively. All the experiments reported in this paper have been carried out using ECJ V.19 [17]. Any parameters not listed here is assumed to be set to the default value provided by the framework.

B. Experiment 1: Minimising Risk

In the first experiment we attempt to find an f for a reconfiguration policy that only cares about risk. In our formulation, this translates into a set of weights $(w_1, w_2, w_3) = (1.0, 0.0, 0.0)$ to be used in the cost function $G(f)$. In a typical experiment, the result is a function that discards nodes’ degree (g) and distance to server (d), and only takes into account the nodes’ local risk (r). Furthermore, the function is always strictly decreasing in r . Some examples of functions found include:

$$f(r, g, d) = \begin{pmatrix} -(\cos r) \\ (\sin r) \end{pmatrix}$$

or the even simpler

$$f(r, g, d) = (* -0.27514 r)$$

This is clearly the expected result: a policy that constantly moves servers to those areas of the network with the lowest risk. The average risk value obtained by these policies is approximately $R = 0.1$, which is extraordinarily low. This, however, comes at a price: the load balance is generally around $B = 0.83$ (a value quite far from being optimal) and the number of relocations is $L = 0.36$ (i.e., relocations occurred 36% of the time).

Fig. 3 (upper plates) shows some snapshots of the effects induced by these policies on the network operation. It is clear how after a few time steps the servers flee from the dangerous areas and accumulate around nodes in the upper-right area. The results for different simulations and at different time steps are equivalent to those shown here.

TABLE I
METRICS OF SOME POLICIES OBTAINED WITH DIFFERENT WEIGHTINGS.

w_1	w_2	w_3	R	B	L
1.00	0.00	0.00	0.10	0.83	0.36
0.00	1.00	0.00	0.54	0.11	0.24
0.00	0.00	1.00	0.56	0.33	0.00
0.33	0.33	0.33	0.56	0.14	0.03
0.40	0.40	0.20	0.33	0.21	0.03
0.50	0.25	0.25	0.18	0.60	0.07

C. Experiment 2: Maximising Performance

Reconfiguration policies that only focus on load balancing can be discovered by using a weighting of the form $(w_1, w_2, w_3) = (0.0, 1.0, 0.0)$. The policy functions obtained are complex and generally difficult to understand analytically (although, as we will see later, they work remarkably well). For example, one search produced the function

$$f(r, g, d) = (- (\cos (\sin (\log (+ (/ r g) (* g d)))))) (\exp (\log (- (- (/ r d) (\sin (* 0.64990 g)) (+ (\cos 0.70840) (/ r 0.64637))))))$$

Some illustrations of the effect of this policy are given in Fig. 3 (middle plates). After some simulation steps, servers tend to migrate to areas where approximately one third of the clients are associated to each server. This constitutes an optimal configuration according to our performance measure. (Note that we are not demanding any centrality measure, hence that some configurations place the servers in peripheral positions where nonetheless the partition into three approximately equal subnetworks is preserved.)

D. Experiment 3: Minimising the Number of Reconfigurations

This experiment produces again an anticipated result. The search converges very quickly to a constant function, such as for example

$$f(r, g, d) = (\exp (/ -0.42278 -0.49757))$$

The effect is obvious: all nodes are equally suitable and, consequently, servers are never relocated. The average number of relocations is therefore $L = 0$, and values for risk and load balancing are approximately $R = 0.56$ and $B = 0.33$.

E. Experiment 4: Combined Objectives

Table I shows the results of some additional experiments where combined objectives are considered, as well as the ones already discussed. For example, when using weights $(w_1, w_2, w_3) = (0.4, 0.4, 0.2)$ we obtained the function

$$f(r, g, d) = (* r (/ (\log (/ (\sin (+ (\log (\exp (\sin (\exp (\log (- d g)))))) (+ (\exp (- (\cos g) (\sin d))) (\sin (* (- (\cos 0.36447) (\cos 0.43138)) (\sin (+ g g)))))) r)) d))$$

Gaining an analytical understanding of what such policies do is not an easy task. In the case shown above, the policy

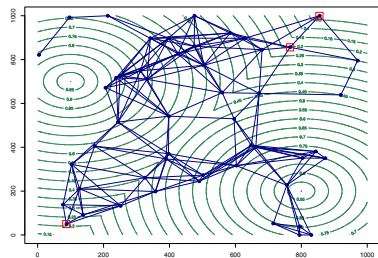
produces an average risk of $R = 0.33$, an average load balancing of $B = 0.21$ and an average number of relocations of $L = 0.03$. While L is very low and R could be considered appropriate, the load balancing is still somewhat high. This can be visualised in Fig. 3 (lower plates). Even in this reduced number of snapshots is obvious that managing risk slightly prevails over load balancing, and the successive policy decisions tend to relocate the servers on suboptimal choices. Such effects may be partially controlled by attempting different weightings in the cost function (see, e.g., Table I). A more adequate approach is presented in the next section, where we obtain a characterisation of the trade-offs among these competing objectives through Multi-Objective Optimisation.

F. Experiment 5: Multi-Objective Optimisation

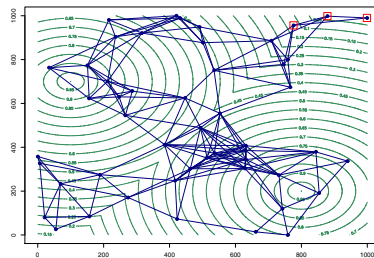
The approach used above implicitly assumes that the policy objectives (i.e., risk, load balance, and number of reconfigurations) are directly comparable and, therefore, can be conveniently weighted and summarised into a single value. The limitations of such a strategy are well known, e.g. what are the best assignment of weights? (i.e., that for which the three goals are simultaneously minimised). In scenarios with certain complexity there are often non-trivial interdependencies between the objectives, in such a way that a good value for one of them automatically implies a not-so-good value for another one. Multi-Objective Optimisation (MOO) algorithms attempt to simultaneously optimise a number of (possibly competing) goals. The main aim is to approximate the Pareto-optimal front, that is, the set of solutions that are not dominated by any other solution. Such a surface reflects the inherent tradeoffs and relationships among the different goals. Furthermore, the set of solutions in the Pareto front is a valuable outcome, as the decision-maker can establish under which circumstances a different tradeoff is required and, consequently, a different solution.

MOO problems naturally arise in all sort of engineering problems. In the case of MANETS, some works have already explored this paradigm (see e.g. [5] for an early discussion of MO routing for tactical aircraft; or, more recently, [8], [16], [18] for a similar application to route-discovery protocols in MANETS). Here we have used Strength Pareto Evolutionary Algorithm (SPEA2) [26] to perform MOO to the reconfiguration policy problem. The obtained Pareto front is a set of solutions (256 in our case, but this is a parameter set by the user) which are essentially incomparable as they provide optimal values for one objective, possibly at the expense of others. We give a preliminary analysis of this set in Fig. 4. The upper plate shows the tradeoffs between R and B for different intervals of L . (Graphs are listed from the bottom and from the left in increasing order of the intervals, so the bottom-left graph corresponds to $L \in [0, 0.04]$, the bottom-middle to $L \in [0.04, 0.09]$ and so on. The amplitudes of the intervals are selected so approximately the same number of solutions are kept in each bin.) It is clearly observed how for high number of relocations ($L > 0.3$), a range of possibilities regarding R and B exists (bottom-left graph). In general, risk and load

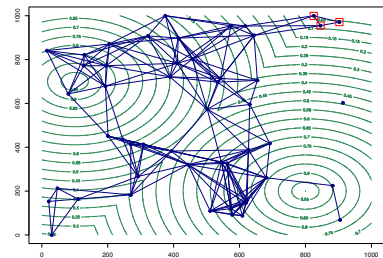
Risk-focused policy – weights: $(w_1, w_2, w_3) = (1.0, 0.0, 0.0)$



$t = 10$

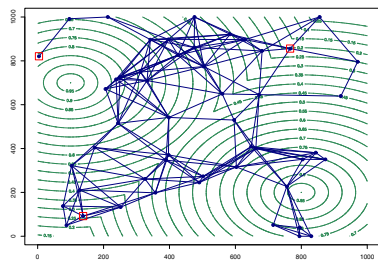


$t = 20$

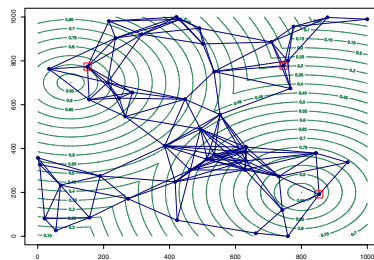


$t = 50$

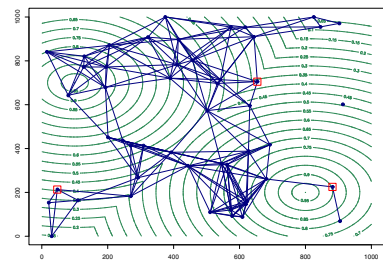
Performance-focused policy – weights: $(w_1, w_2, w_3) = (0.0, 1.0, 0.0)$



$t = 10$

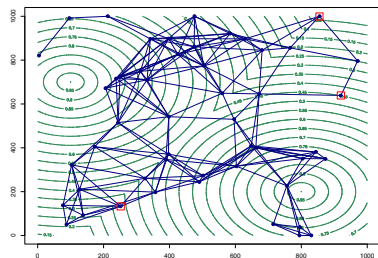


$t = 20$

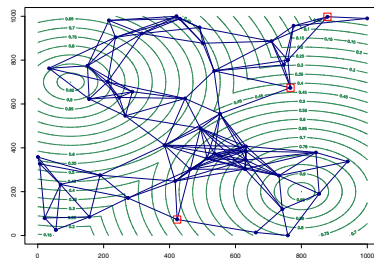


$t = 50$

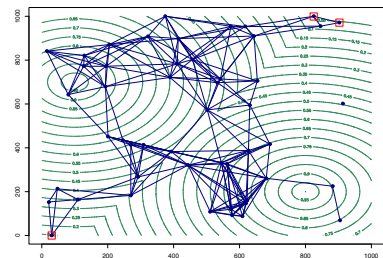
Mixed policy – weights: $(w_1, w_2, w_3) = (0.4, 0.4, 0.2)$



$t = 10$



$t = 20$



$t = 50$

Fig. 3. (In colour in the electronic version.) Placements dictated by three different policies at various time instants. Service providers are marked with a red square. The effect of the different policies is clear even for this reduced number of snapshots.

balance are conflicting goals, so decreasing one of them causes a bad effect on the other. Few relocations translate into pareto fronts where good load balance is difficult to achieve, although risk can be kept in reasonable levels. The middle and bottom plates in Fig. 4 give similar analysis for other comparison of objectives.

VII. CONCLUSIONS

Data mining and combinatorial optimisation techniques have already shown their potential to solve a variety of computer and network security problems. In this work, we have discussed and motivated the need for an additional step in this direction, namely, the *discovery* of security policies for

systems that operate against complex backgrounds where the consequences of a particular policy are not easily foreseeable. We have provided an example for a tactical MANET where sensible services must avoid risk exposure, so a policy must dictate when and how to reconfigure the deployment of such servers. We emphasise that the scenario used through this paper is but an example of the general idea; that is, that traditional ways of producing security policies (specification followed by successive refinement stages) are difficult and very limited. Here we attempt to shift the emphasis away from such an approach towards learning (or searching for) a policy with desirable outcomes.

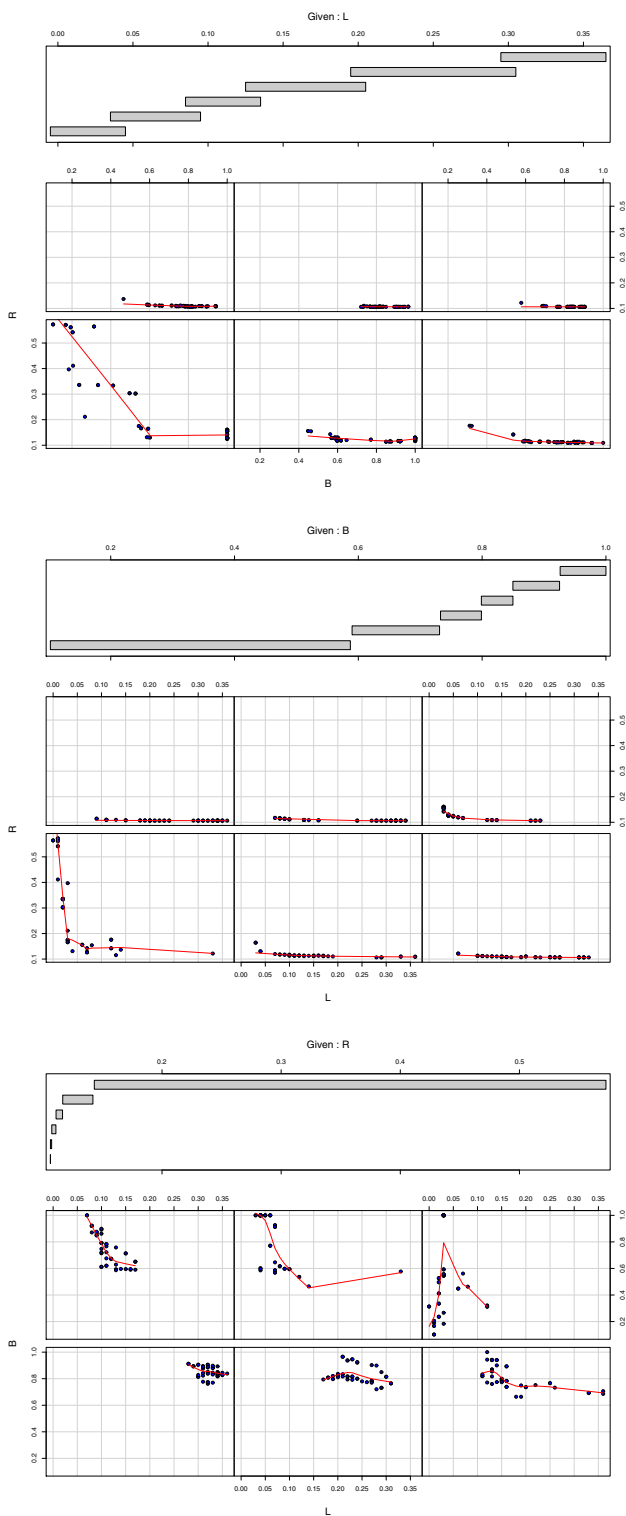


Fig. 4. (In colour in the electronic version.) Pareto fronts.

REFERENCES

[1] P.-C. Chen and P.A. Karger. "Risk Modulating Factors in Risk-based Access Control for Information in a MANET". IBM Research Report RC24494, Feb. 2008.

[2] P.-C. Chen, P. Rohatgi, C. Keser, P.A. Karger, G.M. Wagner, A.S. Reninger. "Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control". *2007 IEEE Symposium on Security and Privacy*, pp. 222–230. IEEE Press, 2007.

[3] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003.

[4] H. Inoue and S. Forrest. "Inferring java security policies through dynamic sandboxing." In *International Conference on Programming Languages and Compilers*, 2005.

[5] E.K. Isensee. "Multicriteria Network Routing of Tactical Aircraft in a Threat Radar Environment." *Master Thesis*. Air Force Institute of Technology, Air University, 1991.

[6] M.C. Jason Program Office. "Horizontal Integration: Broader Access Models for Realizing Information Dominance". *Technical Report JSR-04-132*, The MITRE Corporation, JASON Program Office, Mclean, Virginia, Dec 2004. <http://www.fas.org/irp/agency/dod/jason/classpol.pdf>.

[7] K.A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[8] K. Kotecha and S. Popat. "Multi-Objective Genetic Algorithm-based Adaptive QoS Routing in MANETs." *IEEE Congress on Evolutionary Computation*, pp. 1423–1428, 2007.

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

[10] M. Kuhlmann, D. Shohat, and G. Schimpf. "Role mining – revealing business roles for security administration using data mining technology." In *Proc. SACMAT 2003*, pp. 179–186.

[11] W.B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[12] N. Li, T. Li, I. Molloy, Q. Wang, E. Bertino, S. Calo, and J. Lobo. "Role mining for engineering and optimizing role based access control systems." Purdue University, Tech. Rep. 11 2007.

[13] Y.T. Lim, P.C. Cheng, P. Rohatgi, and J.A. Clark. "MLS Security Policy Evolution with Genetic Programming." In *Proc. GECCO 2008*, pp. 1571–1578.

[14] Y.T. Lim, P. Cheng, J.A. Clark, and P. Rohatgi. "Policy Evolution with Grammatical Evolution." In *Proc. SEAL 2008*, pp. 71–80.

[15] Y.T. Lim, P. Cheng, P. Rohatgi, and J.A. Clark. "Dynamic Security Policy Learning." In *1st ACM Workshop on Information Security Governance*, 2009.

[16] H. Liu, J. Li, Y.-Q. Zhang and Y. Pan "An Adaptive Genetic Fuzzy Multi-path Routing Protocol for Wireless Ad-Hoc Networks." *Proc. 6th Intl. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 468–475, 2005.

[17] S. Luke et al. "ECJ 19 – A Java-based Evolutionary Computation Research System." Available at: <http://cs.gmu.edu/~eclab/projects/ecj>.

[18] S. Marwaha, D. Srinivasan, C.K. Tham and A. Vasilakos. "Evolutionary Fuzzy Multi-Objective Routing for Wireless Mobile Ad Hoc Networks." *Proc. IEEE Congress on Evolutionary Computation*, pp. 1964–1971, 2004.

[19] P. McDaniel. "Policy Evolution: Autonomic Environmental Security." Software Engineering Research Center Showcase, USA, December 2004.

[20] "The Network Simulator – NS-2." <http://www.isi.edu/nsam/ns>

[21] C.E. Perkins, E.M. Royer and S.R. Das. "Ad Hoc On-Demand Distance Vector (AODV) Routing." *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, 1999.

[22] J. Schlegelmilch and U. Steffens. "Role mining with orca." In *Proc. SACMAT 2005*, pp. 168–176.

[23] D. Scott and R. Sharp. "SPECTRE: A tool for inferring, specifying and enforcing web-security policies", 2002.

[24] R.L. Rivest. "S-Expressions." Internet Draft, May 4, 1997. See also <http://people.csail.mit.edu/rivest/sexp.html>

[25] A. Tongaonkar, N. Inamdar, and R. Sekar. "Inferring higher level policies from firewall rules." In *Proc. LISA 2007*, USENIX Association, pp. 1–10.

[26] E. Zitzler, M. Laumanns, and L. Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm." In *Proc. EUROGEN 2001*, pp. 95–100.