A Moving Target Environment for Computer Configurations Using Genetic Algorithms

Michael Crouse and Errin W. Fulp
Department of Computer Science
Wake Forest University
Winston-Salem, North Carolina 27109
Email: fulp@wfu.edu

***Abstract*—Moving Target (MT) environments for computer systems provide security through diversity by changing various system properties that are explicitly defined in the computer configuration. Temporal diversity can be achieved by making periodic configuration changes; however in an infrastructure of multiple similarly purposed computers diversity must also be spatial, ensuring multiple computers do not simultaneously share the same configuration and potential vulnerabilities. Given the number of possible changes and their potential interdependencies discovering computer configurations that are secure, functional, and diverse is challenging.**

This paper describes how a Genetic Algorithm (GA) can be employed to find temporally and spatially diverse secure computer configurations. In the proposed approach a computer configuration is modeled as a chromosome, where an individual configuration setting is a trait or allele. The GA operates by combining multiple chromosomes (configurations) which are tested for feasibility and ranked based on performance which will be measured as resistance to attack. Successive iterations of the GA yield configurations that are often more secure and diverse due to the crossover and mutation processes. Simulations results will demonstrate this approach can provide at MT environment for a large infrastructure of similarly purposed computers by discovering temporally and spatially diverse secure configurations.

I. Introduction

Moving Target (MT) environments provide a type of diversity defense where the attack surface is constantly changing. The intent is that in the time it takes an attacker to perform system reconnaissance and construct an exploit, the system will have changed in such a way the exploit will have a limited impact. Also by employing this dynamic environment, an attacker acting on false, or constantly changing information may expend more resources and have an increased risk of detection.

Different forms of MT defenses have been proposed and successfully used to protect computer systems. Address randomization is a MT defense that changes the location of certain portions of a program's memory [1], [2]. Another MT strategy alters network configurations, such as network routes and address assignments (address hopping) that potentially limits the usefulness of an attacker's reconnaissance [3].

Host-level MT defenses, the focus of this paper, change the computer's configuration, thereby limiting the attacker's knowledge about the system. A computer configuration can be modeled a set of parameters, some of which directly contribute to security (e.g. using `ssh` as opposed to `telnet`) and others that may not (e.g. choice of desktop manager). As with any type of MT strategy, changes need to occur such that the computer configuration is different while remaining functional (provides the services required by the user).

MT strategies can be temporal and/or spatially oriented. A temporal defense seeks to protect a single computer by changing computer properties over time. As a result the lifetime of an exploit might be limited. MT defenses can also be spatial, where the defense seeks to prevent multiple computers from implementing the same configuration at the same time. In the case of an infrastructure of similarly purposed computers, a spatial MT defense could limit the number of computers vulnerable to an exploit at any point in time.

Unfortunately finding secure, functional computer configurations that offer temporal and spatial diversity is problematic. If the configuration parameters are represented as a list of binary values, finding the best configuration can be viewed as a boolean satisfiability problem (referred to as SAT problem) since configuration parameters are not necessarily independent (impact each other). For MT, the problem is further complicated since security of a computer configuration is difficult to quantify [4]. In addition, new security vulnerabilities and the functionalities cause the SAT function to change over time. Fortunately there have been advances in the development of SAT problem solvers [5]. Of these techniques, approaches that iteratively discover possible solutions and introduce some randomness, such as genetic algorithms, are well suited for MT strategies. These search heuristics can find new configurations based on past secure configurations but also have randomness that ensures there are differences on successive configurations [6].

This paper investigates the use of Genetic Algorithms (GAs, a type of search heuristic) to discover new secure and diverse computer configurations. Using this approach a computer configuration is modeled as a chromosome, where individual configuration settings are traits or alleles. The general concept is that good chromosomes are used to generate better chromosomes using a series of selection, crossover, and mutation processes. These processes also incorporate randomness that also can provide diversity. The generated configuration is initially tested for feasibility, which determines if the configuration provides the desired functionality [8]. Another

important component of any GA is determining the fitness of a chromosome (understanding how good the solution is). Although quantifying security in absolute terms is difficult [7], some approaches do exist to provide security evaluation of a configuration [4]. For the proposed approach, the security of a configuration will be scored based on the number of security incidents that occurred for a duration of time, where fewer incidents over longer periods of time are considered better. Therefore the fitness is determined after the generated configuration has been implemented for a certain amount of time. The process repeats to maintain temporal diversity.

Simulation results will show this proposed system could provide a MT environment for an infrastructure of computers. In these experiments configurations are periodically changed and the security of the system and diversity of the configurations are measured. Results show that an infrastructure, where all computers start with the same poor configuration, is able to improve security while maintaining a temporal and spatial diversity. Furthermore the impact of imperfect security knowledge is investigated.

The remainder of this paper is structured as follows. Section II describes computer configurations and the difficulty of managing security. The proposed genetic-algorithm based method for creating a moving target defense is described in Section III. Section IV empirically analyzes the effectiveness of the approach, while section V summarizes the paper and discusses some areas of future work.

II. Computer Configurations and Moving Target

A computer configuration is a set of parameters that govern the computer's operation. This can include transient data, such as information found in the Linux `/proc` file system, and persistent data that is stored in non-volatile memory such as the Windows registry and Unix resource files. Collectively this is a large amount of information that is often distributed across a number of files and databases [9]. For example the Windows registry has over 77,000 entries initially, but after loading a common suite of applications can have on average 200,000 settings [10].

How the parameters are set impacts the system security by enabling or disabling certain exploits. For example the Federal Desktop Core Configuration settings and the Consensus Security Configuration Checklist managed by NIST provides guidelines and configuration settings for various operating systems [11]. Many of the configuration parameters individually impact security, for example using `ssh` instead of `telnet` for remote access. However it is possible that configuration settings are interdependent therefore a combination of settings is required to improve security while implementing just one change does not. For example several configuration settings may be required to secure a web server. Other configuration parameters may have no known impact on security.

Configurations must also be feasible, providing the user the necessary computer functionality. For example a configuration that denies any network access may be secure but is infeasible for a web server. Considerable research has been done to develop methods that can identify configuration anomalies [8] and assessing security [7]. However determining a functional and secure configuration (best possible parameter settings) still remains problematic given the large number of possible configuration parameters and parameter interdependency.

A. Difficulty of Discovering Secure Configurations

Discovering secure computer configurations can be shown to be difficult in theory. Let a computer configuration C be modeled as a collection of n parameters, $C = \{p_1, p_2, ..., p_n\}$. Furthermore let each configuration parameter p_i be a binary value. Note configuration parameters often have a range of possible values (consider the POSIX permissions for a file), however it would be possible to model these values as multiple binary parameters.

Suppose given C there is a subset $C' \subset C$ that impacts security. Assume there are multiple vulnerabilities to protect against and one functional, secure configuration. Furthermore assume that parameter interdependencies are possible as described in the preceding section. As a result the configuration C' can be viewed as a boolean expression written using only AND, OR, NOT, variables (configuration parameters), and parentheses. For example OR would be used to denote multiple configuration parameters that address the same vulnerability, while parenthetical ANDs would represent interdependent parameters. As such the configuration expression can be expressed in Conjunctive Normal form (CNF).

The problem is to determine the parameter values that results in the boolean expression equivalent of C' to be true (all vulnerabilities have been addressed). The difficulty of this problem can be understood by reducing variants of the satisfiability problem (SAT), specifically circuit satisfiable (CIRCUIT-SAT) [12]. Therefore determining the set of parameters that satisfies the configuration expression can be viewed as $\mathcal{NP}$-complete as well. Determining secure configurations is more problematic since the expression is not always known. For example the boolean expression would change over time, once new applications are installed and new vulnerabilities are discovered.

B. Configurations and Moving Target

As previously described, a host-level Moving Target (MT) environment seeks to change the configuration of a computer to provide temporal and spatial diversity. Temporal diversity refers to the configuration differences of one computer over time, which limits the amount of time an attacker can use an exploit. Spatial diversity ensures that multiple computers do not share the same configuration at any point in time. As a result an exploit can be launched against a smaller number of computers. Considering the boolean expression equivalent of configuration C, moving target seeks solutions (if they exist) that satisfy the expression. All the candidate configurations can share the same C' configuration settings since they address vulnerabilities but the remaining configuration parameters should vary to provide temporally and spatially diverse configurations.

Given the mapping between configuration discovery and SAT problems, it is possible to leverage SAT algorithms to provide a MT environment. These solutions can be categorized into two broad categories, backtracking-oriented algorithms and stochastic local search algorithms. A backtracking approach, such as the Davis-Putnam-Logemann-Loveland (DPLL) algorithm, select a literal in the expression, assigns a truth value, then simplifies the expression then recursively check if the resulting expression is satisfiable. Stochastic search algorithms, such as genetic algorithms, search the parameter space and attempt to find a better solution based on previous good solutions. In addition genetic algorithms have been used to find solutions when there is no or limited knowledge of the specific structure of the problem instances which is also the case for MT since the configuration expression can change over time [6]. Therefore genetic algorithms are well suited for discovering configurations that provide a MT environment.

III. A Genetic Algorithm for Creating a Moving Target Environment

Genetic Algorithms (GAs) are a type of search heuristic that attempts to mimic evolution. As with most evolutionary algorithms, the general concept is to find better (more fit) solutions by discovering, emphasizing, and recombining good building blocks – portions of the solution that yield better solutions [6]. When applied to configuration management, these building blocks are part of the C' set of configuration settings. However note, the objective of a MT environment is not to find the best configuration, rather to discover and periodically implement a set of secure configurations (feasible configurations that share C' settings, while differing otherwise).

Before the GA can be applied to a problem, a genetic representation of the solution domain, determination of feasibility, and an understanding of fitness is needed. GAs represent potential solutions as a chromosome that consists of multiple traits, or parts of the solution. Considering the individual configuration settings of a computer system as traits, a complete configuration can be modeled as a chromosome. As previously discussed, configuration traits can have a range of values, representing different configuration settings. For example a binary valued trait can be used to represent the existence of a file or if address space randomization is enabled, or a trait can have a larger range of possible values such as which TCP implementation is active (i.e. Tahoe, Reno, Vegas, etc...).

For computer configurations, a chromosome is considered feasible if the resulting configuration provides the necessary functionality for the computer. For example, if the computer is a web server then any configuration that blocks network access would be deemed infeasible and not added to the pool. While some feasibility tests can be performed programmatically (e.g. network connectivity), others may require human input to determine if the performance of the system sufficient.

Fitness is a measure of how good the solution is as defined by the chromosome; for MT a chromosome is considered fit if the corresponding configuration is secure. Unfortunately it is impossible to know if a configuration is secure *a priori* since new vulnerabilities will be discovered over time. Considering how secure systems are administered can provide a methodology for determining fitness. Current configuration management practices involve implementing to a certain degree of security and making modifications once a security violation occurs. Using this general idea, a chromosome's fitness will be based on the number and type of security incidents discovered over time; of course the hope is that the configurations will incur no security incidences. For example consider information security, which concerns protecting the confidentiality, integrity and availability of information (often referred to as the CIA model). Incidents can be classified based on whether the confidentiality, integrity, or availability of information was impacted. Based on the classification different weights can be associated with the incident, for example an incident that only involves availability could be considered less problematic that integrity or confidentiality.

Unfortunately just because a configuration incurred no security incidences does not necessary mean the configuration is secure. It is possible that the system was simply not targeted. As a result the GA must cope with some uncertainty in regards to fitness which will be explored experimentally in Section IV. Finally, the security of a configuration will also be based on the the last time it was implemented. Older configurations, which are possibly outdated, will be considered less fit than recently implemented configurations.

A. Genetic Algorithm Processes

Although GA's have been researched since the 1960's, there is not a rigorous definition that clearly differentiates a GA from other evolutionary based strategies [6]. However, most GA implementations use some combination of *selection*, *crossover*, and *mutation* to search the solution space. This is depicted in Figure 1. These three operators are defined such that information is not lost as the operators are applied, ensuring over time solutions will improve.

Selection identifies a portion of the chromosome pool used to generate a new chromosome. For example it is possible to associate a higher likelihood of selection to more fit chromosomes. The hope is that chromosomes with the highest fitness will produce even more fit offspring. It is also possible to allow a more non-dominating approach in order to produce more of a mix of solutions. For configuration generation, selection was implemented using a Monte Carlo method, where higher weights were associated with better chromosomes. This approach to selection is often referred to as "elitism" can typically improve the GA's performance [6], [13].

Possibly the most distinguishing feature of a GA is the crossover process [6] which is the combination of multiple chromosomes (chosen from the selection process) to form a new chromosome. Therefore a new configuration is created by randomly combining portions of existing configurations. For example consider a crossover process that produces one child chromosome from two parents. The one-point crossover

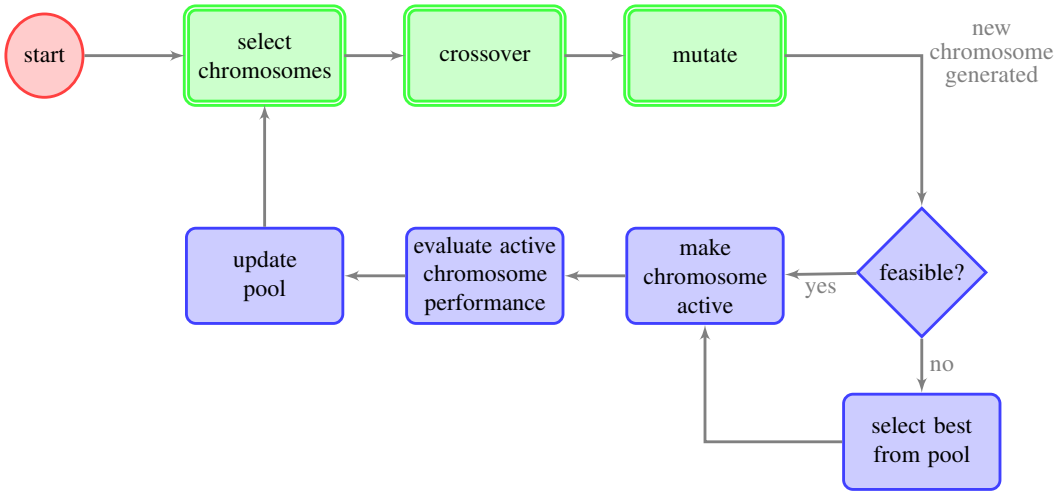

Fig. 1. Flow chart of the tasks associated with the proposed GA-based MT environment. Double lined blocks (also in green) contain *traditional* GA processes.

identifies a single point, perhaps randomly, in the chromosome where the traits beyond that point are traded to produce the child. While simple to implement, a single-point crossover is generally not effective given long chromosomes and can suffer from positional bias where portions of a good solution are lost [6], [13]. In contrast multiple-point crossover that uses more that one crossover point is better in this situation and is also better for discovering computer configurations.

The last process is mutation, which randomly changes settings in the offspring chromosome. The purpose of this process is to maintain diversity across the generations of chromosomes and avoid permanent fixation at any particular locus. As applied to MT mutation will also help provide diversity, which will be demonstrated in the experimental result section. A simple example of mutation is to randomly select a trait or set of traits, then randomly select a new value for each.

B. Using a Genetic Algorithm for Configuration Management

When a GA is used to solve an optimization problem, completing an iteration of the three operators results in a new set of chromosomes, referred to as a new generation or offspring. These chromosomes are measured for feasibility and fitness, the chromosome pool is updated and then the process repeats until the successive generations converge to an answer. Although similar in some regards, this will **not** be the approach followed when a GA is used to provide a MT environment.

The complete set and order of processes associated with the proposed MT approach is given in figure 1. Initially the process starts with a minimum set, or pool, of feasible chromosomes (configurations). From the pool the two best chromosomes are chosen using the selection process. The crossover process is then performed on the chosen chromosomes, which creates a new chromosome. The new chromosome then undergoes the mutation process, which randomly changes some of the traits (configuration parameters within feasibility). The new chromosome then undergoes a feasibility test, and perhaps a simple security audit, to ensure that it will provide the necessary functionality. If feasible (which may require an administrator's oversight), the new chromosome is placed into service, which will be referred to as the active chromosome. The active chromosome stays in service for a period of time so to obtain insight into its fitness (measured as the number of security incidences). The amount of time can be static or dynamic, depending on the implementation.

Once the service period is complete, the active chromosome is given a security score indicating how it performed and is used to update the chromosome pool. If the chromosome pool is not full, then the active chromosome is added. If the pool is full, then the active chromosome can replace the weakest chromosome if it is better. The process then repeats as seen in Figure 1. Therefore the GA is allowed to advance one iteration for each new chromosome generated.

IV. Experimental Results

An initial performance investigation of the proposed MT environment was conducted using simulation. Each experiment simulated an infrastructure of 256 similarly purposed computers, which is indicative of an office or college laboratory of workstations. This size infrastructure and set of simulated computers allows the measurement the spatial and temporal diversity achieved over time.

The configuration for each simulated machine consisted of 80 parameters (settings), where half of the parameters in the configuration were identified to impact security. Each parameter was a binary value, therefore the solution space consisted of 2^{80} possible, but perhaps not feasible, configurations. In addition, certain groupings the security parameters were considered interdependent; therefore, a certain pattern of configuration settings were necessary to improve security

in these cases, as described in section II-A. The remaining parameters did not impact security, however certain settings may be required to support functionality (recall this would impact feasibility). Each computer in the infrastructure was given the same initial configuration that was susceptible to all the vulnerabilities. Therefore initially the infrastructure had zero spatial diversity and computers had the worst possible configuration settings in terms of security.

As described in Section III-A, the GA has a few important variables required for its proper operation, specifically the chromosome pool size, and probabilities of crossover and mutation. The chromosome pool size was 10, while the crossover and mutation rates were 0.8 and 0.02 respectively. These values were determined empirically (common for GA's [6], [13]) to maintain a balance of diversity and security. The GA per simulated computer operated with no knowledge of which parameter impacted security. Once the GA produced a configuration its feasibility was determined (determine if a minimum set of parameters were set). If feasible, the configuration was placed in service on the simulated computer. This configuration will be referred to as the active configuration, which stays in operation until the next iteration of the GA.

For each experiment, the diversity and security of the simulated computers were recorded over time. Diversity was either spatial or temporal, and was measured using a Hamming distance treating the configuration as an 80 dimension vector. Temporal diversity was the Hamming distance between successive configurations, while spatial diversity was measured as the average Hamming distance between any two active configurations in the infrastructure. Therefore as the Hamming distance increases the diversity (temporal or spatial) also increases. Security was measured as the number of vulnerabilities associated with the active configuration, which will be referred to as the vulnerability score. Therefore a zero vulnerability score is desired.

A. Temporal and Spatial Diversity

Figure 2 shows the temporal diversity for one computer using the proposed MT approach. Note time is measured in iterations, since a new configuration is generated per iteration of the GA. Again diversity is measured by taking the Hamming distance between the current active configuration and the previous active configuration. Given the number of configuration parameters, the maximum diversity was equal to 80 while the minimum was zero (indicating successive configurations were identical). The graph shows the configuration changed rapidly from its initial settings. The distance between successive configurations then reduced and averaged approximately 8 per iteration for the remainder of the simulation. Note this distance can be viewed the number of parameter differences between successive configurations. This indicates that secure distributions were found that varied in terms of the remaining non-security related parameters. This is caused solely by the genetic operations being performed, crossover and mutation.

Figure 3 shows the average spatial diversity across all computers in the infrastructure. In this Figure the Hamming distance is the minimum distance measured between all possible unique computer pairs. The Figure shows the each GA quickly found configurations that were different from each other. However once the GAs found secure configurations, at approximately 90 iterations, the Hamming distance reduces. This distance reduction occurs when the configurations differ primarily by the non-security related parameter settings (configuration share the same required secure settings); therefore the search space for the GA has reduced. The spatial distance is approximately 21 for the remainder of the simulation, indicating the approach provides a significant level of temporal and spatial diversity.

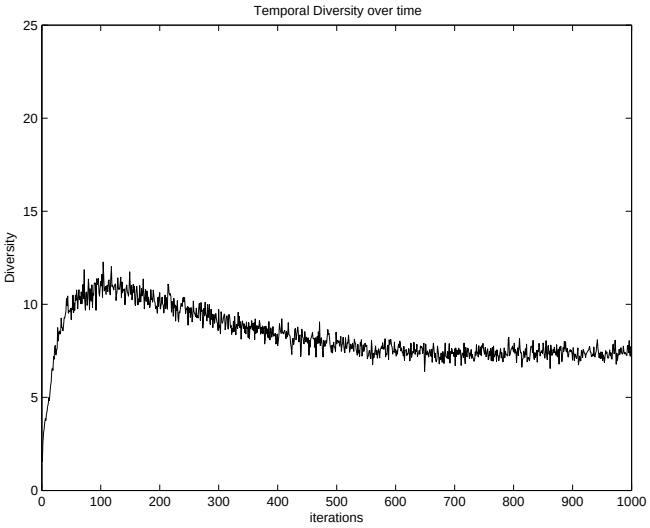

Fig. 2. Graph of the average temporal diversity of one computer in the infrastructure.

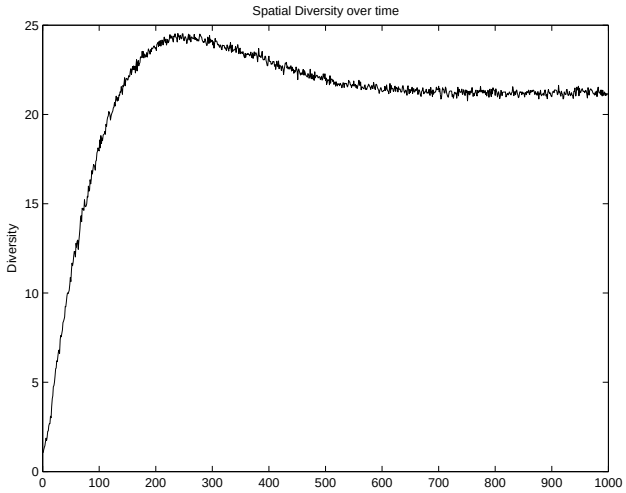

Fig. 3. Graph of the average spatial diversity of the infrastructure of 256 computers.

B. Configuration Vulnerabilities

Although the objective of a MT approach is not to find the best possible configuration, the proposed system should

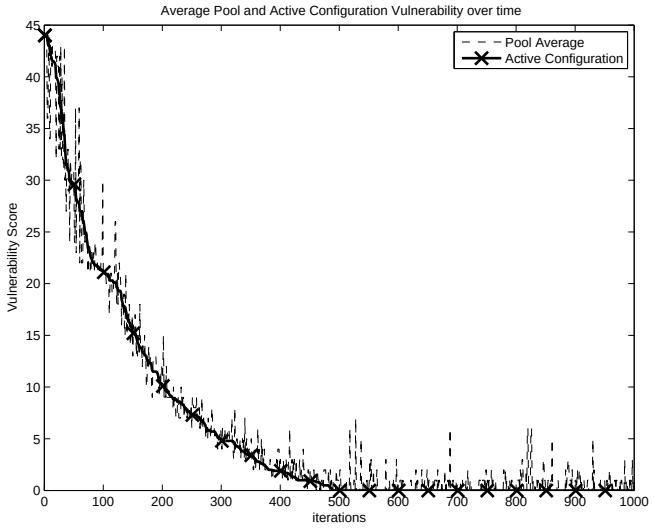

Fig. 4. Graph of the vulnerability scores of a computer using the proposed genetic algorithm with 0.05 mutation rate.

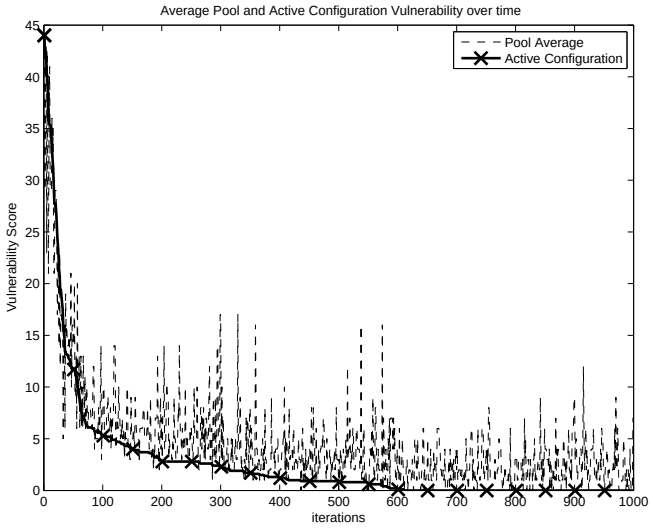

Fig. 5. Graph of the vulnerability scores of a computer using the proposed genetic algorithm with 0.2 mutation rate.

provide secure configurations per algorithm iteration. Again, security will be measured as the number of vulnerabilities associated with a configuration and will be referred to as a vulnerability score; therefore a zero vulnerability score is desired. Two values were recorded for each simulated computer. First, the active vulnerability score is the number of vulnerabilities associated with the configuration currently implemented on the computer and will change over time as new configurations are implemented. Second, the average pool vulnerability score is the average number of vulnerabilities associated with the configurations in the chromosome pool (10 configurations per pool per computer). This value indicates the overall health (security) of the configurations used by the GA. It is hoped that both vulnerability scores will approach as time progresses.

Figure 4 shows the vulnerability scores of a simulated computer in the infrastructure. Although only the results of a single computer are presented, they are representative of all the computers in the infrastructure. As seen in the figure, the active configuration score started high since the original configuration consisted of multiple vulnerabilities; however approached zero as the number of iterations increased. After 150 iterations the GA produced active configurations that had approximately half the number of vulnerabilities, while at 500 iterations the active configuration had very few, if any, vulnerabilities. The active configuration value does vary per iteration, which is due to the GA processes, specifically mutation. The pool vulnerability score similarly starts high and moves towards zero as the number of iterations increases. Therefore the vulnerability score decrease indicates the GA is maintaining a healthier set of configurations as time progresses.

As mentioned in the previous paragraph, the mutation rate has an impact on the performance of the GA. Recall mutation is the last GA process for the proposed method, as seen in figure 1. When performed, portions of the configuration are randomly changed to prevent the GA from converging to local minima. Therefore increasing the mutation rate can help the GA find better solutions more quickly; however it can also be accompanied by higher variability. This is demonstrated in Figure 5 which depicts the vulnerability scores of a simulated computer where mutation rate is 0.2 (originally 0.05). The benefit of a higher mutation rate can been seen in the pool vulnerability score, which reduces significantly faster than the previous example (Figure 5). The vulnerability score is halved in approximately 20 iterations as opposed to 150 in the previous simulation. The GA is able to find better configurations more quickly, causing better pool vulnerability scores. The active vulnerability score also trends downward, however it is accompanied by a much larger variability.

C. Perceived Configuration Vulnerabilities

Although it is possible to determine the actual security of the configuration in this simulated environment, it is impossible to know exactly in reality, as described in section III. To mimic the uncertainty of the security evaluation, these experiments altered the vulnerability score by a Normal distribution with average of 5. Experiments will measure the impact of uncertainty associated with vulnerability scores; however note, perturbing the vulnerability score had little effect on diversity.

Figure 6 shows the vulnerability scores (actual value) for a simulated computer in the infrastructure, where the mutation process had a 0.05 mutation rate. Similar to the previous experiments, the scores decreased as the number of iterations (time) increased, however at 500 iterations the pool vulnerability score stabilizes at approximately 4. This is due to the uncertainty of the scores used by the GA. The active score follows the same trend, decreasing until approximately 500 iterations where the average value stabilized. Therefore uncertainty with regards to the security of a configuration

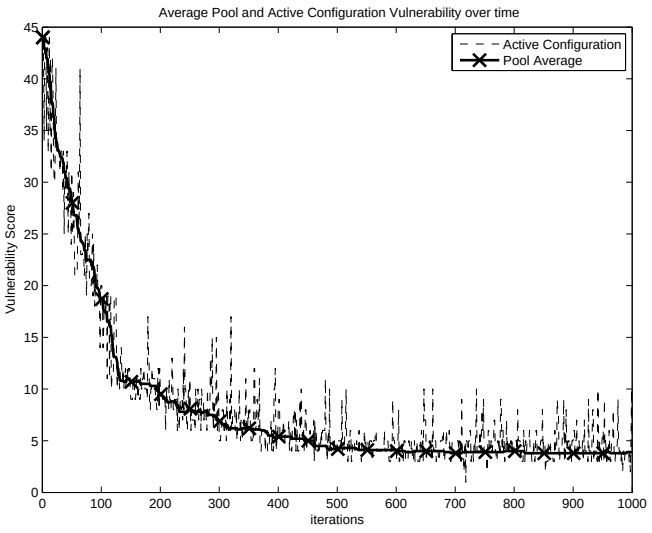

Fig. 6. Graph of vulnerability scores of a computer using the proposed genetic algorithm with 0.05 mutation rate.

does impact the GA performance, however the same effect is observed in reality since administrators must often act on imperfect (incomplete) information. This impact will diminish over time since a better understanding of prior configurations is typically gained (analysis in hindsight).

V. Conclusions and Future Work

Moving Target (MT) environments provide security by changing the attack surface, thereby limiting the amount of time an attack is valid. The diversity sought can be either temporal or spatial. Temporal diversity represents the change over time, while spatial diversity is measured as the difference between any two systems at any particular time. For computer systems this is accomplished by changing system properties that are defined in the configuration. However given the number of potential configuration parameters and their interdependence, determining alternative diverse configurations that are secure and feasible (provides the necessary functionality) is problematic.

This paper introduced an MT approach that uses a Genetic Algorithm (GA) to find temporally and spatially diverse secure configurations. A computer configuration is modeled as a chromosome, where each configuration parameter is considered a trait or allele. The GA processes the chromosomes (potential configurations) using a series of selection, crossover, and mutations operations. The intent is that better configurations will be found using the previously good configurations as inputs. The random components of the GA, specifically crossover and mutation, provide a higher likelihood that the configurations are diverse. The performance of the approach was demonstrated by simulating a network of 256 similarly purposed computers. The experimental results demonstrated that the approach was able to achieve both temporal and spatial diversity. Starting with vulnerable configurations, the approach found configurations that were more secure and diverse. The speed by which configuration are discovered can be improved by altering the mutation rate, but also results in higher variability. The impact of imperfect knowledge about a configuration was also investigated. It was shown that altering the perceived security does effect the configurations generated, however it is expected that this will have a limited impact in the future (rating of configurations should improve as time passes).

While the experimental results indicate the proposed GA-based MT environment has potential, more research is need to better understand how the system operates in different situations. For example more research is needed to understand the appropriate GA crossover and mutation rates. The amount of time required to rate a configuration needs additional research. Furthermore more research is needed to understand the impact on spatial diversity as the number of computers increases.

References

[1] S. Forrest, A. Somayaji, and D. H. Ackley, "Building diverse computer systems," in *Proceedings of The 6th Workshop Hot Topics in Operating Systems*, 1997.

[2] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks," in *In Proceedings of the 7th USENIX Security Symposium*, 1998, pp. 63–78.

[3] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagostakis, "Defending against hitlist worms using network address space randomization," *Computer Networks*, vol. 51, pp. 3471–3490, 2007.

[4] S. Stolfo, S. M. Bellovin, and D. Evans, "Measuring security," *IEEE Security and Privacy*, vol. 9, pp. 60–65, 2011.

[5] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," in *Handbook of Knowledge Representation*. Elsevier, 2008, vol. 3, ch. 2, pp. 89 – 134.

[6] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[7] M. Montanari and R. H. Campbell, "Multi-aspect security configuration assessment," in *Proceedings of the 2nd ACM workshop on Assurable and usable security configuration*, 2009, pp. 1–6.

[8] E. Kiciman and Y.-M. Wang, "Discovering correctness constraints for self-management of system configuration," in *In Proceedings of the 1st International Conference on Autonomic Computing (ICAC04)*, 2004.

[9] Z. Huang, "Automatically identifying configuration files," Master's thesis, Electrical and Computer Engineering, University of Tornoto, 2009.

[10] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang, "Strider: A black-box, state-based approach to change and configuration management and support," in *Proceedings of the 17th USENIX conference on System administration*, 2003, pp. 159–172.

[11] NIST, "Consensus security configuration checklist," http://web.nvd.nist.gov/view/ncp/repository.

[12] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

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