Improving the Diversity Defense of Genetic Algorithm-Based Moving Target Approaches

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

***Abstract*—One approach for providing a Moving Target (MT) defense is to intermediately change a system's configuration (operating systems and/or applications). For example, Genetic Algorithms (GAs) have been successfully used to find alternative configurations that can discount the attacker's knowledge about the system. Central to the GA approach is the chromosome pool, which consists of the best alternative configurations discovered thus far. Unfortunately the pool can possibly "stagnate" if these configurations do not change after a period of time. Although the configurations are secure, this situation limits the diversity the approach can achieve.**

This paper describes how chromosome pool management can improve the diversity of GA-based MT environments. The proposed approach "ages" configurations, reducing the fitness (security) of a configuration based on the period of time since it was last active (used as the configuration for the system). As a result, configurations that not been active for a long period of time are considered less secure which can make space in the pool for new alternatives. Simulations results will demonstrate proper pool management can provide a functional, secure, and more diverse MT environment.

I. Introduction

Reconnaissance is a critical phase of most cyber attacks, where the attacker seeks to gain intelligence about potential targets. Kewley et al. [1] suggested that a well-resourced adversary (i.e., a nation state) spends approximately 45 percent of their time performing reconnaissance. Therefore given the expense associated with reconnaissance and the reliance on this knowledge for a successful attack, neutralizing this attack phase can be a successful defense strategy [2].

A Moving Target (MT) defense is a way to disrupt the reconnaissance phase of a cyber attack. MT environments provide a diversity defense where the attack surface constantly changes. In the time it takes an attacker to perform system reconnaissance and construct an exploit, the system will have changed so that the exploit will have or appear to have limited impact. In addition, an attacker acting on false or constantly changing information may expend more resources and increase the risk of detection. Note diversity can be measured either temporally or spatially. 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.

In [3] a new MT defense was introduced that intermittently changed a system's configuration (operating systems and/or applications). Using a Genetic Algorithm (GA) and pool of configurations encoded as chromosomes, new configurations are discovered using a series of selection, crossover, and mutation processes. Using these processes, the approach seeks to discover better solutions based on previously known solutions. As applied to MT environments, fitness equates to the security of a configuration. Once a configuration is discovered, feasibility and preliminary security testing is performed to ensure the configuration provides the necessary functionality and security. Simulation results showed the configurations were secure and temporally and spatially diverse [3]. However this paper will show that the chromosome pool can potentially become stagnate, where after a number of GA iterations the same configurations are always present. This situation limits the new configurations that can be discovered, which negatively impacts the diversity of the system.

This paper refines the GA process of this MT defense by investigating and improving chromosome (configuration) pool management. Specifically the fitness (security) associated with configurations are aged by lowering scores based on the period of time they were last made active. As a result, older configurations in the

pool "age-out" and make room for new configurations. This approach follows the notion that configurations once deemed secure may become less secure as new vulnerabilities are discovered. Simulations will show that the proposed pool management technique will improve the configuration diversity, with little negative effect on the fitness of the configurations.

The remainder of this paper is structured as follows. Section II reviews various forms of MT defenses. The genetic-algorithm based method for creating a moving target defense is described in Section III. Configuration pool management is reviewed in section IV. Section V empirically analyzes the effectiveness of the approach, while section VI summarizes the paper and discusses some areas of future work.

II. Different Moving Target Approaches

MT defenses have been successfully used to protect computer systems at different levels. Address randomization is a MT defense that changes the location of certain portions of a program's memory. For example varying the location of stack components is a common MT defense that can successfully prevent many forms of overflow exploits [4], [5].

Another MT strategy alters network configurations (routes or address mappings) to limit the usefulness of an attacker's reconnaissance [6], [7]. For example address shuffling remaps the addresses assigned to computers after certain intervals of time. As a result the attacker cannot be certain that an IP address will always contact the same computer. While this level of MT has been shown to be somewhat effective, it requires coordination of several resources to ensure legitimate users are not adversely affected, which is difficult to achieve.

Host-level MT defenses change a computer's appearance over time. For example attackers often attempt to determine the Operating System (OS) of a remote computer since this knowledge can help identify potential vulnerabilities and exploits [8]. TCP/IP fingerprint obfuscators prevent attackers from sensing the actual host OS by portraying different OS characteristics (changing the response to probe packets). If the OS appearance is changed after various periods of time then a MT defense can be provided [9]. However various methods have been developed to identify the actual OS of a computer even if an obfuscator is used [10].

As an alternative to the previously described host-level MT defenses, this paper focuses on a technique that directly manipulates the computer configurations [3]. This approach proactively discovers secure configurations or variations of one secure configuration and places them in service at varying periods of time. Unlike obfuscators, it can be applied to the OS and/or applications. The approach not only provides a MT defense but also moves the system toward a more secure posture, which is especially important as new functionalities are introduced and vulnerabilities discovered over time.

III. Using Genetic Algorithms to Create a Moving Target Defense

This paper seeks to improve a host-level Moving Target (MT) environment that leverages evolution-based searching algorithms to find diverse, secure configurations [3]. Evolution-based search techniques have been successfully used in various ill-conditioned and dynamic environments. Computer configuration management and discovery is a similarly ill-conditioned environment, since configuration parameter interdependencies may not be completely understood, as well as new functionalities and threats will appear over time [3].

By mimicking processes found in nature, this technique has the ability to develop better solutions by combining previously known good solutions. Therefore this approach is also well suited for discovering multiple good computer configurations. Furthermore, as done in [11], the approach can use Virtual Machines (VMs) to test the security and feasibility of newly generated configurations (burn-in period) thereby limiting the risk to the actual hosts enacting generated configurations.

A. Mapping Configurations to Chromosomes

A computer configuration is a set of parameters that collectively govern its operation. This can include transient data 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 [12]. 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 [13]. While these parameters control how the operating system and applications perform, they also affect security [14].

GAs represent potential solutions as a chromosome that consists of multiple traits, or parts of the solution. A computer configuration can be modeled as a chromosome if the individual configuration settings are considered traits. 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, cubic, etc...). Given a chromosome, some configuration parameters may impact the system security, while others may affect diversity. Ideally it is best to have variety parameters that provide both, resulting in multiple secure configurations that can be used to provide a moving target defense. A reverse mapping from the chromosome traits to the actual configuration parameters is also needed to enact a chromosome.

B. Feasibility and Fitness

To be considered a potential solution, a chromosome must be feasible and have a certain fitness, or measure of goodness. For computer configurations, a chromosome is considered feasible if the configuration it represents provides the necessary functionality for the computer as defined by the user and/or administrator. For example, if the computer is a web server, then any configuration that blocks network access would be infeasible and not enacted. Previous research has focused on identifying and correcting misconfigurations [15]–[17].

For this MT system, fitness corresponds to configuration security. Existing resources and guidelines help to identify basic configuration security concerns [18], [19], but the security status/level/ranking of a configuration is never fixed; new vulnerabilities will be discovered. Current computer and network configuration management practices first implement a certain degree of security and make modifications once a violation occurs. A similar process has been used to determine the functionality and performance of configurations [20]. Using this general approach, a Virtual Machine (VM) and can be used to actuate the configuration translated from the chromosome. An initial assessment of the configuration can be done using penetration testing, where the success and failures of different attack vectors are used to score the configuration. For example the *vulnerability score* used in the experimental section of this paper counts the number of illegal accesses of a computer resource; therefore a low vulnerability score is desired. If the chromosome is later made active on the actual host, feedback regarding its live performance will be used to update the fitness measure. This score will not change until the configuration is made active again. If the time between activity is long then accuracy of the score should be discounted since new vulnerabilities may be discovered during this period of time. This fitness decay process is explored further in Section IV.

C. Genetic Algorithm Processes

Although GAs have been extensively researched since the 1960s, no rigorous definition clearly differentiates a GA from other evolution-based strategies [21]. However, most GA implementations use some combination of *selection*, *crossover*, and *mutation* to search the solution space, which is depicted in Figure 1.

Selection identifies a portion of the chromosome pool (set of candidate chromosomes) 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. Possibly the most distinguishing feature of a GA is the crossover process [21] 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 identifies a single point, perhaps randomly, in the chromosome where the traits beyond that point are traded to produce the child. 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.

D. Genetic Algorithms and Configuration Management

Figure 1 shows the complete set and order of processes associated with the proposed MT approach. The approach can be divided into two major parts: discovering new configurations using the GA and managing configurations on the actual host for an MT environment.

For the configuration discovery portion, the process initially starts with a minimum set, or pool, of feasible chromosomes (configurations). Chromosomes are chosen among the best in the pool using the selection process. The crossover process is then performed on the chosen chromosomes, which creates a new chromosome. The mutation process is then applied, which randomly changes some of the traits (configuration parameters within their respective ranges). The new chromosome then undergoes an initial feasibility test, and perhaps a simple security audit to ensure that it will provide the minimal functionality and security. This can be accomplished by combining good security guidelines and techniques already developed for misconfiguration

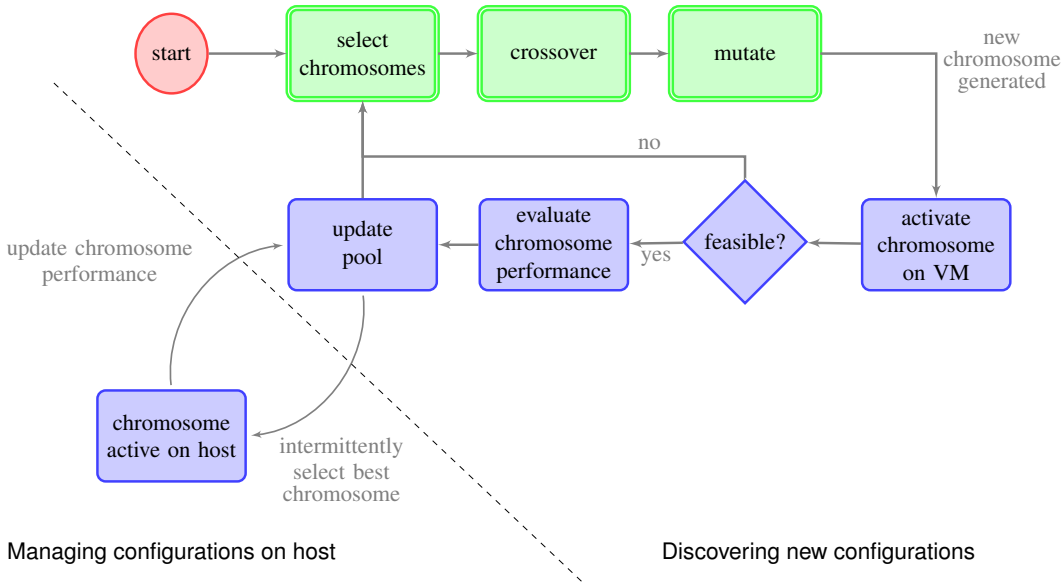

Fig. 1. Flow diagram of the tasks associated with the GA-based MT environment. Double-lined blocks (also in green) contain the *traditional* GA processes.

management. If feasible, the new chromosome is placed into service using a Virtual Machine (VM) that duplicates the applications of the actual host. This active chromosome undergoes further automated functionality and penetration testing to obtain insight to its fitness (measured as the number of security incidences).

Once the testing is complete, the active chromosome is given a vulnerability score indicating how it performed and the chromosome pool is updated. 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, but only one iteration for each new chromosome generated. The approach is proactive since it will constantly look for new configurations. Converging to a single solution is not the objective since security will change as new vulnerabilities are discovered, rather the objective is the creation and management of a good set of potential configurations (the chromosome pool) that forms the basis of our MT defense. If the GA does converge to a single chromosome then it suggests that all the parameters in the chromosome affected security, and none are available for diversity. If the chromosome stored only a sub-set of the possible configuration parameters, then configuration parameters could be added to the chromosome to increase the configurations search space. A combination of security and diversity configuration parameters is ideal.

The MT portion of the system is responsible for changing the configuration of the actual host based on GA findings. Intermittently, the MT process will select a configuration from among the most fit in the chromosome pool. The time during which a configuration is active should vary to counter possible attacks. At the end of the interval, the configurations performance is reassessed; the fitness revised; and the chromosome pool updated. However there is the potential that the pool of configurations will not change after some period of time, which limits the diversity of the active configurations.

A prototype has been developed to manage the configuration of a Debian-installed server that is intended to provide remote access to legitimate users. This Python-based system can read and set various configuration parameters (e.g. file permissions) as directed by the GA. As previously described feasibility is based on maintaining a certain set of functionality (e.g. ssh access) while fitness of the active configuration is determined based on how well the confidentiality, integrity, and availability of certain resources are maintained.

IV. Chromosome Pool Management

As described in the previous section, the MT approach maintains a pool of the best chromosomes encountered thus far, where the fitness is determined based on testing

(predicate, activation on a VM, and activation on the actual system). However the confidence in this score should change over time. The fitness should decrease as the period of time since the configuration was active increases. This should be done because what is considered secure today may not be in the future, which is due to the discovery of new vulnerabilities.

When fitness scores of the chromosomes do not change over time the pool can become stagnate. In this case the chromosomes present in the pool remain the same, and as a result the GA operates with the limited set of candidates. Although these configurations are perceived to be secure (hence their strong fitness values), always using the same chromosome pool will potentially limit the new configurations that will be discovered on subsequent GA iterations. This limits the effectiveness of the MT strategy and the discovery of better configurations.

One simple approach to prevent stagnation is to increase the mutation rate (the final process of the GA). However this causes the GA to operate more as a random search since a larger portion of the parameters is just randomly changed. As result few settings from previous good solutions survive the GA processes, and future solutions do not build upon previous good solutions. Another approach is to increase the pool size; however, since the current selection process only uses the best two configurations increasing the pool size would not affect the performance.

This paper proposes *decaying* the chromosome's fitness scores over time. This process reduces the fitness score by a value β after each iteration of the GA. As a result less used configurations will have a lower fitness and be replaced by newer configurations, which creates a desired amount of churn. If a configuration is secure, then it should be selected, made active, and the score will be updated which will allow it to survive multiple iterations. The described function is linear, but exponential functions were also considered, for example doubling the decay amount per iteration. However these approaches were found to be too aggressive, resulting in the GA behaving as a random search.

V. Experimental Results

In this section the performance of not managing the chromosome pool (as done in [3]), and decaying at different rates is examined empirically using simulation. The simulated environment consisted of 100 similarly purposed machines, representing an office of workstations. Each chromosome, representing a computer's complete configuration, consisted of 80 alleles or configuration parameters. Half of these configuration parameters have some impact on the system's security and the other parameters contribute to the diversity. Each parameter is a binary value providing a solution space of 2^{80} possible configurations. As stated in Section III-A, configuration parameters can be interdependent in terms of their impact on security. For these experiments, several parameters were grouped together to form dependencies and the fitness function takes this detail into account. Each simulated computer is given the same initial configuration in which is susceptible to all vulnerabilities. As a result the initial diversity is zero and the computers have the worst possible fitness score.

As described in III-C, there are several GA parameters that must be specified and impact the search capabilities. These parameters include the chromosome pool size, and the probabilities for cross over and mutation. Previous research explored potential values and settled on 10 chromosomes for the pool size with a crossover rate of 0.8 and 0.02 mutation rates [3], and will also be used for these experiments.

For each experiment the diversity, security, and the average age of the chromosomes within each pool was recorded over time (measured as iterations of the GA). The diversity, both temporal and spatial, was calculated using the Hamming Distance. Temporal is calculated for each machine, comparing the previous active configuration to the newly generated one. The spatial diversity was calculated after each iteration and is the average minimum distance between all machines within the simulated set of machines. In this case greater Hamming distances between computers indicates a greater diversity.

The security of a configuration was calculated based on the number of vulnerabilities associated with the given configuration, which will be referred to as the vulnerability score. Therefore lower scores are desired, where a score of zero indicates a configuration has no vulnerabilities. If fitness decay is used, then after each iteration the fitness scores for configurations in the pool are changed as described in Section IV. Given fitness is measured as a vulnerability score for these experiments (lower scored are desired), each vulnerability score is increased by β. Two different β values were considered for these experiments: increasing the score by 1 every 10 iterations ($\beta = 1/10$) and increasing the score by 1 every 50 iterations ($\beta = 1/50$).

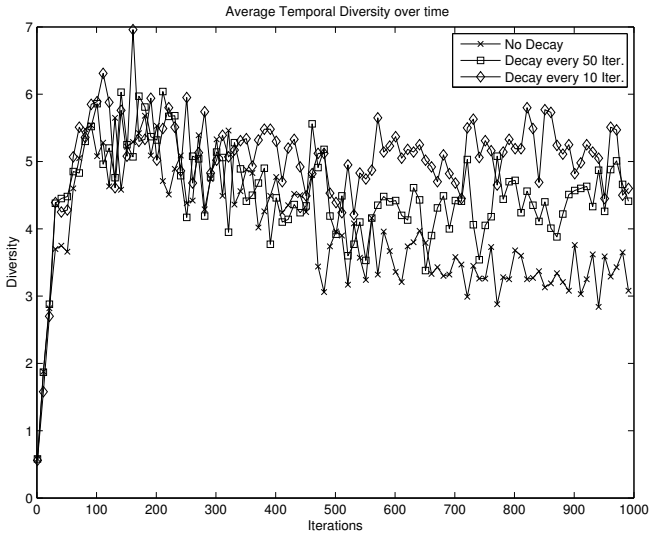

Fig. 2. Average temporal diversity of one computer in the infrastructure with varying decay rates.

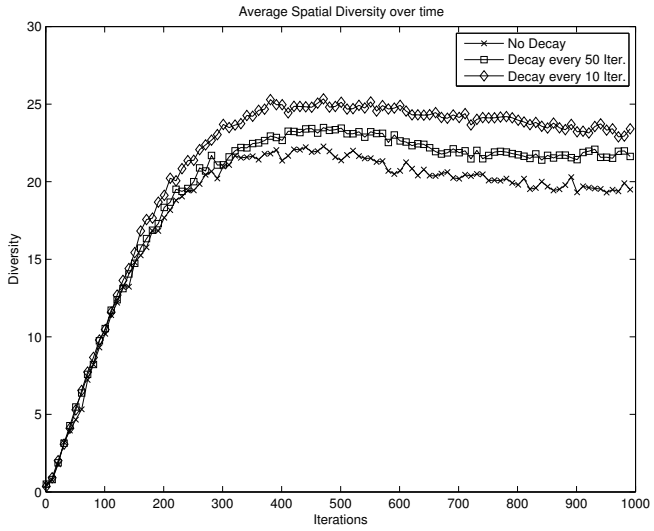

Fig. 3. Average spatial diversity of the infrastructure with varying decay rates.

A. Temporal and Spatial Diversity

Figure 2 shows the average temporal diversity for the computers using the GA-based MT approach, where diversity is the Hamming distance between successive active configurations. Note, time is measured in iterations since a new configuration is generated per iteration of the GA. The graph shows the configuration changed rapidly from its initial settings. The distance between successive configurations then reduced, where the highest decay ($\beta = 1/10$) rate maintained the highest temporal diversity, averaging approximately a distance of 5. In contrast, no pool management (or no score decay) resulted in the lowest diversity measurement, averaging a distance of 3. These distances 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; however higher decay rates help maintain a higher diversity.

Figure 3 shows the spatial diversity for all the computers in the infrastructure. For these measurements, diversity is the minimum Hamming distance 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 300 iterations, the Hamming distance starts to stabilize. 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 is reduced. However decaying the pool provides greater diversity, where the highest decay rate (decay every 10 iterations) performed best. Similar to the temporal diversity results, no decay performed the worst providing the lowest diversity. These results indicate that managing the chromosome pool can significantly improve the diversity of the configurations discovered.

B. Configuration Vulnerabilities

The primary focus of a MT environment is to produce both temporally and spatially diverse configurations. However, the approach should also provide secure configurations. If diversity were the only metric, performing a random search would be the optimal approach. 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.

Figure 4 shows the average vulnerability score of the active configurations in the infrastructure. As seen in the figure the vulnerability scores steadily decrease as time increases; approaching zero after 800 iterations. The decay rate does affect the vulnerability score, where the higher decay rate has a slight increase as compared to no decay. Therefore a higher decay rate can cause good configurations to be removed from the pool, although there is a minimal impact on the vulnerability score.

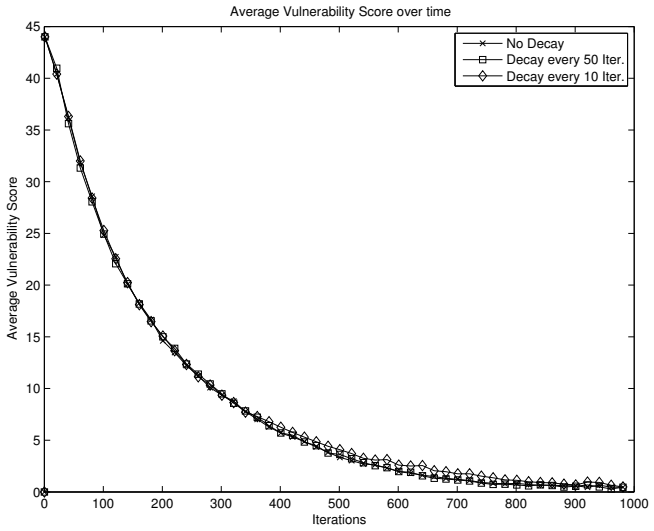

Fig. 4. Graph of average vulnerability scores of active configurations in the infrastructure.

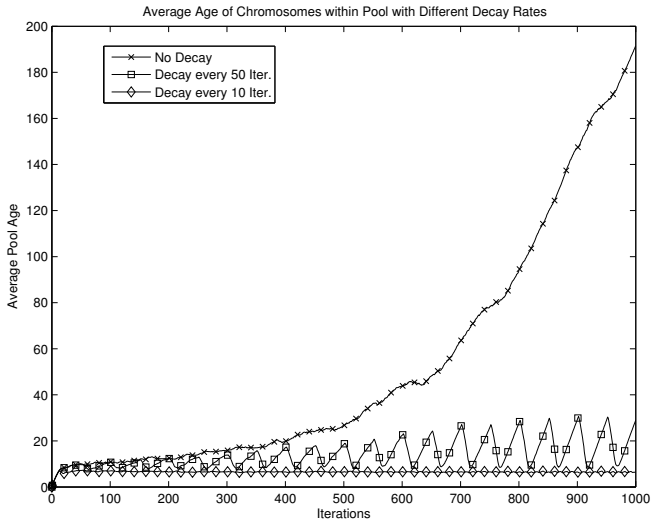

Fig. 5. Graph of average age of the chromosome in the pools for the infrastructure.

C. Average Pool Age

The age of a configuration is number iterations a configuration is either active or resident within the pool. Each configuration, if feasible, is only active for a single iteration therefore the youngest possible configuration within the pool is one iteration. Figure 5 shows the average pool age for one computer using different decay rates. At the beginning the average pool age remains relatively low regardless of how often the pool is decayed. However as the GA approaches the most fit configurations, the decay rate impacts the average pool age. If the pool is decayed every 10 iterations, the average age of the pools remains very low. This is beneficial, as more of the newly generated configurations will be added to the pool because the old configurations are considered less secure. This is shown in the diversity results. In contrast, if no decay is used then the pool eventually stagnates and the pool consists of the same configurations. If the decay is every 50 iterations then the pool achieves a higher age before returning to a lower average.

D. Pool Diversity

Another metric to consider when discussing pool management is the diversity within the pool. If the pool, representing the set of possible secure configurations, is not diverse, the new solutions will be less diverse. Figure 6 shows the pool diversity over time with no decay, and decay every 10 or 50 iterations of the GA. When the chromosomes within the pool are not decayed over time, the diversity begins to decrease where the

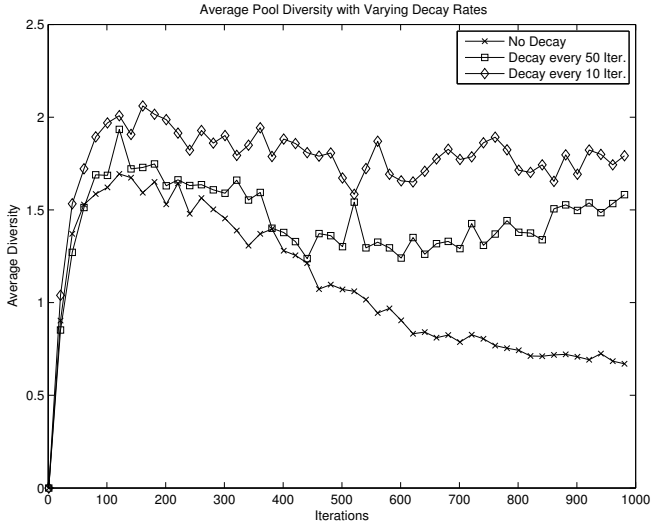

Fig. 6. Graph of the average pool diversity for the computer infrastructure.

average distance between solutions within the pool is less than one. Therefore decaying the pool slightly after several iterations can improve the diversity of the pool. The diversity within the pool will help ensure that the GA produces more diverse solutions both spatially and temporally, which was observed in the previous section.

VI. Conclusions and Future Work

Moving Target (MT) environments can invalidate an attacker's reconnaissance by changing the attack surface. 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 can be accomplished by changing system properties that are defined in the configuration. For example it is possible to use a Genetic Algorithm (GA) to create a MT environment. Using this approach configurations are encoded as chromosomes and a series of selection, crossover, and mutation processes are used to discover secure and diverse configurations.

A critical component of the MT approach is the chromosome (configuration) pool, which stores the best configurations discovered thus far. It is possible for the pool to stagnate which results in the pool always consisting of the same configurations. Although these configurations are secure, limiting the pool can impact the diversity. This paper described how pool management can be used to avoid pool stagnation. Specifically, the fitness (perceived configuration security) should decay based on the amount of time since it was made active. As a result, less recently used configurations will be considered less secure, which will potentially make for newly discovered configurations. The experimental results showed decaying the fitness of configuration in the pool does help the diversity of configurations. For example higher decay rates maintained higher temporal, spatial, and pool diversity over time. Decaying the fitness to increase diversity did not negatively impact security; the fitness of the active configurations was observed to be approximately equal.

While these and previous result show using a GA to create a MT environment has promise, more research is needed further understand how the approach operates under different situations. For example more research is needed to understand how the system will react to changing security requirements. In addition, more research is needed to understand the impact on spatial diversity as the number of computers increases.

REFERENCES

[1] D. Kewley, R. Fink, J. Lowry, and M. Dean, "Dynamic approaches to thwart adversary intelligence gathering," in *Proc. of the DARPA Information Survivability Conference & Exposition II (DISCEX '01)*, vol. 1, 2001, pp. 176–185.

[2] J. F. Dunnigan and A. A. Nofi, *Victory and Deceit: Deception and Trickery at War*, 2nd ed. San Jose, California, USA: Writers Club Press, 2001.

[3] M. B. Crouse and E. W. Fulp, "A moving target environment for computer configurations using genetic algorithms," in *Proceedings of the 4th Symposium on Configuration Analytics and Automation (SafeConfig 2011)*, 2011.

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

[5] 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.

[6] 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.

[7] J. Michalski, C. Price, E. Stanton, E. Lee, K. S. Chua, Y. H. Wong, and C. P. Tan, "Final report for the network security mechanisms utilizing network address translation LDRD project," Sandia National Laboratory, SAND Rep. SAND2002-3613, Nov. 2002.

[8] M. Smart, G. R. Malan, and F. Jahanian, "Defeating tcp/ip stack fingerprinting," in *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, 2000.

[9] E. Al-Shaer, *Moving Target Defense: Advances in Information Security*. Springer, 2011, ch. 9, pp. 153–159.

[10] G. Prigent, F. Vichot, and F. Harrouet, "Ipmorph: fingerprinting spoofing unification," *Journal in Computer Virology*, vol. 6, pp. 329–342, 2010.

[11] L. L. Wu, G. E. Kaiser, J. Nieh, and C. Murphy, "Deux: Autonomic testing system for operating system upgrades," Columbia University, Department of Computer Science, Tech. Rep., 2008.

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

[13] 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.

[14] J. Oberheide, E. Cooke, and F. Jahanian, "If it ain't broke, don't fix it: challenges and new directions for inferring the impact of software patches," in *Proceedings of the 12th conference on Hot topics in operating systems*, 2009.

[15] A. Whitaker, R. S. Cox, and S. D. Gribble, "Configuration debugging as search: finding the needle in the haystack," in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, 2004.

[16] E. Kycyman, "Discovering correctness constraints for self-management of system configuration," in *Proceedings of the First International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 28–35.

[17] 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," *Science of Computer Programming*, vol. 53, no. 2, pp. 143 – 164, 2004.

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

[19] C. for Internet Security, "Security and assessment benchmark-tools," http://www.cisecurity.org/resources-publications/.

[20] Y.-Y. Su, M. Attariyan, and J. Flinn, "Autobash: improving configuration management with operating system causality analysis," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 237–250, Oct. 2007.

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