Evolutionary Based Moving Target Cyber Defense

David J. John
djj@wfu.edu

Robert W. Smith
smitrw12@wfu.edu

William H. Turkett
turketwh@wfu.edu

Daniel A. Cañas
canas@wfu.edu

Errin W. Fulp
fulp@wfu.edu

Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109, USA

ABSTRACT

A Moving Target (MT) defense constantly changes a system's attack surface, in an attempt to limit the usefulness of the reconnaissance the attacker has collected. One approach to this defense strategy is to intermittently change a system's configuration. These changes must maintain functionality and security, while also being diverse. Finding suitable configuration changes that form a MT defense is challenging. There are potentially a large number of individual configurations' settings to consider, without a full understanding of the settings' interdependencies.

Evolution-based algorithms, which formulate better solutions from good solutions, can be used to create a MT defense. New configurations are created based on the security of previous configurations and can be periodically implemented to change the system's attack surface. This approach not only has the ability to discover new, more secure configurations, but is also proactive and resilient since it can continually adapt to the current environment in a fashion similar to systems found in nature.

This article presents and compares two genetic algorithms to create a MT defense. The primary difference between the two is based on their approaches to mutation. One mutates values, and the other modifies the domains from which values are chosen.

Categories and Subject Descriptors

I.2.8 [**Problem solving, Control Methods and Search**]: Heuristic methods—*genetic algorithms*; K.6.5 [**Security and Protection**]: unauthorized access

Keywords

computer security; moving target defense; directed mutation

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.2598394.2605437 .

1. INTRODUCTION

Cyber attacks commonly initiate with a reconnaissance phase, where the adversary attempts to learn as much as possible about the potential target before launching an attack. This intelligence is very valuable, and it has been reported that a well-resourced adversary will spend approximately 45 percent of its time on this preliminary task [10]. Given the investment associated with reconnaissance and the reliance on this knowledge for a successful attack, neutralizing this attack phase has been shown to be a very successful defense strategy [7].

A Moving Target (MT) defense is a way to disrupt the attacker's attempt to learn how a system operates during the reconnaissance phase of a cyber attack. MT environments provide a diversity defense where the attack surface changes constantly. In the time it takes an attacker to perform system reconnaissance and construct an exploit, the system changes. As a consequence of the modification, the exploit will have limited impact or cause the attacker to lose confidence in the reconnoiter. In addition, an attacker acting on false or constantly changing information may expend more resources and increase the risk of being detected.

MT defenses have been successfully employed to protect computer systems at different levels. For example, address randomization is a MT defense that changes the location of certain portions of a program's memory [4, 9]. While widely implemented, this application-level defense only mitigates a certain type of attack. Another MT strategy alters network configurations to limit the usefulness of an attacker's reconnaissance [1, 14]. This MT defense operates at the infrastructure level and is somewhat effective, but it requires coordination of several resources to protect legitimate users, which is difficult. Host-level MT defenses operate at the system level and change the computer's appearance over time; however, current implementations simply cloak the system which is essentially security through obscurity.

This article describes a new type of multiple host-level MT defense that changes both the host and network attack surfaces. It does this by manipulating the settings of multiple computer's configurations directly. Using evolution-inspired techniques, the approach proactively discovers multiple functional and secure configurations and then places them in service across multiple computers during varying periods of time. As a result, the attacker faces an unpredictable computing environment which can limit the useful-

Parameter	Value Type	Description
`.htpasswd`	Binary	Allow configuration changes on a per-directory basis
`ServerTokens`	Value from a list	Controls the information about server OS and modules.
`KeepAlive`	Binary	Allow multiple requests to be sent over the same connection.
`KeepAliveTimeout`	Positive integer	Time to wait for a subsequent request before closing the connection.
`FollowSymLinks`	Binary	Allow symbolic links in a directory to be followed.
`IncludesNoExec`	Binary	Allow server side include execution.
`Indexes`	Binary	Allow automatic directory indexing.
`LimitRequestBody`	Positive integer	Limit the size of message body.
`LimitRequestFields`	Positive integer	Limit number of request HTTP header fields allowed.
`LimitRequestFieldSize`	Positive integer	Limit the size of an HTTP request header field.
`LimitRequestLine`	Positive integer	Limit the size of a client's HTTP request-line.
`autoindex_module`	Binary	Allow icons for directory listings.
`AllowOverride`	Value from a list	Controls if earlier configuration directives can be overridden.
`LimitExcept`	Value from a list	Limit request methods.

Table 1: A sample of the Apache configuration parameters used in the MT defense system. Parameter names, settings types, and brief descriptions for these 14 are shown.

ness of any reconnaissance activities. Unfortunately, finding configurations that meet these requirements is challenging. There are potentially a large number of individual configurations settings to consider, and for each one of them an administrator may not know that some are interdependent. It is also difficult to quantify the security of a computer configuration [18] since the introduction of new security vulnerabilities and functionalities can make a once secure configuration insecure.

However, evolution-based algorithms, which formulate better solutions from good solutions, are well suited for addressing such ill-formed and dynamic problems. Similar to nature, selection, crossover, and mutation processes are used over successive generations to find better solutions, or configurations, for this host-level MT system. As well, specially designed mutation operators can be developed for this system. As a result, this evolution-inspired MT approach can find secure and diverse configurations over time with appropriate feedback. This will be demonstrated with 102 parameters, taken from Apache web-server (Apache) on Red Hat Enterprise Linux 5 (RHEL5) configurations, that becomes more secure over time, while configurations maintain a level of diversity across the generations to support multiple feasible system configurations.

2. COMPUTER CONFIGURATIONS

A computer configuration is a set of parameters that govern operation. This includes operating system settings and information found in application configuration files. Furthermore, configurations can include transient data, such as the Linux `/proc` file system, or persistent data such as that found in Unix resource files. Collectively, this configuration information is a large amount of data that is often distributed across a number of files and databases.

While these parameters control how the operating system and applications perform, they also affect security. For instance, the Federal Desktop Core Configuration settings and the Consensus Security Configuration Checklist managed by NIST (National Institute for Standards and Technology) and CIS (Center for Internet Security) provide guidelines and configuration settings for various applications and operating systems [8, 16]. For example, Table 1 provides 14 Apache configuration parameters, value types, and parameter descriptions found in this checklist. Note the variety of parameter value types, which include binary, integer, and enumerated values. Furthermore, parameters can be interdependent, so a combination of settings is required to improve security [11]. For example, the NIST guidelines for Apache 2.2 identifies 4 settings to secure the `ScoreBoardFile` directive, which supports interprocess communication. Of course, other configuration parameters may have unknown impacts on security.

Although configuration guidelines can help to improve system security, they may not apply to all computer installations. A system may need certain parameter settings to remain functional, even though those settings may be deemed insecure. In addition, future security threats may render current guidelines useless. Therefore, while intermediately making configuration changes can provide a MT defense by rendering an attacker's reconnaissance ineffective, searching the large number of parameter values to find functional and secure changes is very challenging [6].

3. MOVING TARGET DEFENSE

Although searching for secure, functional, and diverse configurations is difficult, evolution-based search algorithms are well suited for these types of problem environments. Genetic Algorithms (GAs) are a type of search heuristic that naively mimic evolution [15]. Like most evolutionary algorithms, they seek better (more fit) configurations by discovering, recombining, and altering portions of current configurations to generate new ones. However, an MT environment does

not aim to find a single best configuration but to ascertain and periodically implement a configuration from a set of assorted secure configurations. Before a GA can be applied to the MT problem, a genetic representation of the problem domain, methods of determining feasibility, an understanding of configuration fitness, and the design of GA operators must be carefully addressed.

3.1 Configurations as Chromosomes

In nature, an organism has a set of rules that define how it is created. These rules are encoded in genes that are connected together into long strings called chromosomes. Each gene contributes to the traits of the organism, like eye color or height, and has several different settings. Genes and their settings are usually referred to as an organism's genotype. The phenotype is the physical expression of the genotype, which is the organism itself.

In this MT defense system, GAs represent the current configuration generation as a set of chromosomes, each of which consists of multiple configuration parameter settings. Each genotype specifies a phenotypic description, though this relationship may be many to one. Consider the 14 configuration parameters shown in Table 1. There are multiple possible settings for each parameter. Therefore, each MT chromosome in this GA represents a configuration, and each generation consists of multiple MT chromosomes.

It is not necessary for the chromosomes to represent all possible configuration parameters. It may be advantageous to initially limit the number of parameters for the purposes of testing to better understand how they affect security and diversity. Over time parameters can be added to increase the search space and to find more complex configuration settings.

To be considered a potential suitable configuration, a chromosome must be feasible. 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 be enacted. Future implementations will incorporate comprehensive functionality testing, which is discussed in Section 6.

3.2 Chromosome Fitness

Chromosome fitness measures the level of security the configuration provides. While it is difficult to quantify the security of a configuration directly, it can be inferred based on the number of security events detected during operation. The evaluation of the parameter setting is based on the effect it may have on information security, specifically the effect on confidentiality, integrity, and assurance, also referred to as the CIA model [3]. Confidentiality refers to the disclosure of information to unauthorized individuals, while integrity refers to the accuracy and consistency of information, and assurance refers to the availability of information and/or services.

The security score also takes into account the difficulty of exploiting the parameter's vulnerability and the impact if the associated exploit is performed. These items can then be combined to create a Common Vulnerability Scoring System (CVSS) score: a vector of categorical scores associated with the 6 criteria of *access vector* (AV), *access complexity* (AC), *authentication* (AU), *confidentiality* (C), *integrity* (I), and *assurance* (A).

Ideally, attack information about a configured running system can be gathered, analyzed and then interpreted as chromosome fitness. However, this type of information is at best incomplete, and is likely to contain many false positives. For this research, a CVSS based rubric is applied to each parameter setting which culminates into a chromosome fitness score. This fitness score approximates knowledge on the uncertain quantity of attack information.

3.3 Genetic Algorithms

Genetic Algorithms (GAs) are a heuristic, naively mimicking evolution, to search complex or ill-formed problem spaces. GAs have been extensively researched since the 1960s, and most GA implementations use some combination of of the selection, crossover, and mutation operators [15]. These processes are among the tasks associated with the MT defense, as seen in Fig. 1. For the MT environment developed thus far, the search space is a subset of possible Apache and RHEL5 configurations, each consisting of 102 parameters. For many organisms, the selection of parents is the initial step in the creation of a child, and this sampling occurs in such a way as to bias the next generation towards the more fit individuals in the current one. The resultant offspring may have a portion of the genes from one parent and the remainder from the other. This process, crossover, encourages the propagation of discovered traits from the current generation into the next. In addition, some of the offspring's genes may be mutated, randomly changing the gene's value. The importance of mutation is its potential to introduce new traits into future generations. Using these three basic processes, selection, crossover and mutation, organisms can adapt over time.

Selection identifies members of the current chromosome pool as parents for new chromosomes. Roulette wheel selection, also known as fitness proportional selection, uses a probability distribution directly related to chromosome fitness to choose parents. The aim is to encourage chromosomes with higher fitnesses to produce even more fit offspring, while still allowing for offspring from a lesser fit parent. Tournament selection choses 4 chromosomes, has two single elimination rounds of comparisons to determine a winner. In the first round, two winners are determined through two pair comparisons, and in the last round, an overall chromosome is selected by choosing the best of the two first round winners. Tournament selection is less likely to pick chromosomes with smaller fitness values.

Possibly the most distinguishing feature of a GA is the crossover process, the combination of parent chromosomes to form a new chromosome. Therefore, a new configuration is created by combining portions of existing configurations. The crossover process implemented for this simulation is known as *two point crossover*, which has an associated probability p_c. After selecting a parent chromosome in the current generation, p_c is the probability that another chromosome from the current generation is selected and their genetic information exchanged. If two point crossover is not applied then the immediate offspring is the first parent; otherwise, two locations along a random permutation of the genes of the two selected chromosomes are identified randomly, and the genes between the two loci are traded to produce a child.

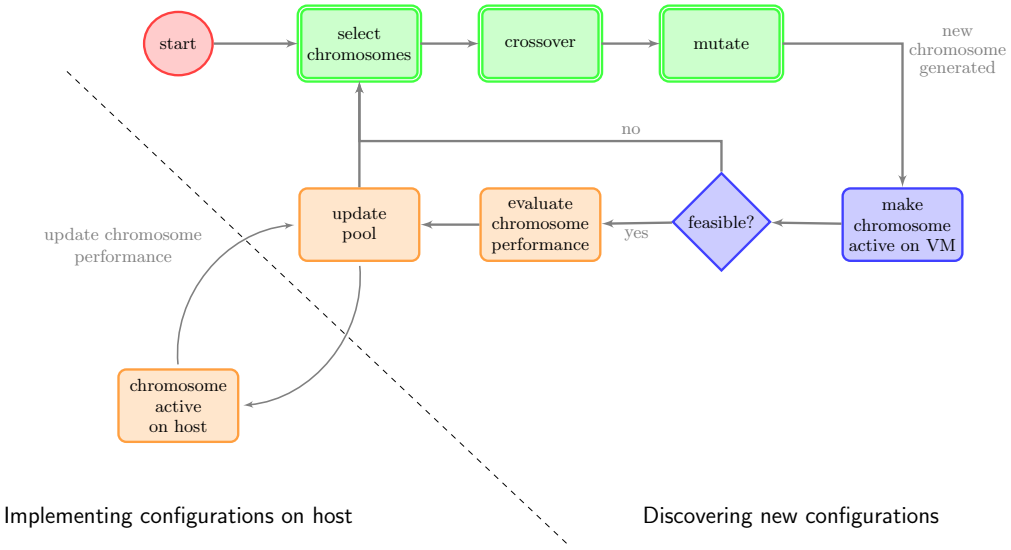


Figure 1: Flow chart of the tasks associated with the GA-based MT environment. Double-lined blocks (also in green) contain *traditional* GA processes, blocks in blue are associated with the virtual machine (VM) component, and blocks in orange are part of the assessment component.

The last process is mutation, which randomly changes settings in the offspring chromosomes. Associated with the mutation operator is a probability, p_m. The purposes of the mutation process are to maintain diversity, to allow extensive exploration of the chromosomes across the generations, and to avoid permanent fixation at any particular locus. With a probability of p_m, mutation is applied to a randomly chosen gene of a chromosome. If the rate of mutation is set too high, the resulting GA is often no better than mere random search. Mutation is not a uniform operator across the configuration parameters. Instead, it differs on a case by case basis between different parameters, according to their type and prior knowledge of acceptable or common settings. In fact, there are two mutation operators: one which applies simple changes to a parameter's value, and the other which makes changes to a parameter's domain. These two mutation operators result in two searching algorithms: the genetic algorithm with parameter value mutation (GA+PVM), and the genetic algorithm with parameter domain mutation (GA+PDM).

The parameter value mutation operator modifies the current value of a chosen chromosome's parameter based on its type. For instance, a single option which may have only one out of a number of possible values would be mutated by changing to a different option. Meanwhile a series of options, each of which may independently be turned on or off, may be modeled as a bit vector representing which options are on. Mutation would then flip a bit. Other parameters may take an integer value. In this case, mutation would randomly select a new number based on a probability distribution which takes into account which range of values the administrator believes is likely to contain secure settings.

The parameter domain mutation operator, a form of directed mutation [2], when given a chosen parameter, uses machine learning techniques to modify the domain of that parameter. If a particular parameter setting has historically been correlated with fitness decreases when adopted, or fitness increases when abandoned, the machine learning algorithm will deem it insecure. This directed mutation operator then removes that value from the parameter's domain. Additionally, if the parameter happens to be one in which a distance function can quantitatively compare settings' similarity, such as bit vectors or numbers, a support vector machine [5] is used to calculate a region of the domain surrounding known insecure values, and this entire subset of values is removed. Once a domain modification is made, then all chromosomes in the current generation must be adapted to comply with the mutated domain. The motivations for this more aggressive operator are to take advantage of the fitness function's convexity to increase the rate of fitness gain, to reduce the number of low fitness individuals in a generation, and to export information about the GA's function across generations.

The success of this MT strategy depends upon the evolution of diverse populations which are increasingly more secure. The genetic algorithm operators naturally tend to move towards a more homogeneous, less diverse, population. An adaptive selection operator, a diversity based blend of Tournament and Roulette Wheel selection, assists in keeping the populations more diverse.

The first generation of MT chromosomes is initialized randomly. Creating the next generation begins with the selection operator choosing parent chromosomes from the current chromosome pool. The crossover operator is applied to the chosen chromosomes, producing the child chromosomes. The mutation operator is then applied to each of these children, which possibly changes some of the chromosome traits (configuration parameters).

4. MOVING TARGET SYSTEM

Prior simulations have demonstrated the effectiveness of using evolution-based algorithms to discover secure and diverse configurations [6]. Based on these encouraging results, a Python-based prototype MT system has been developed

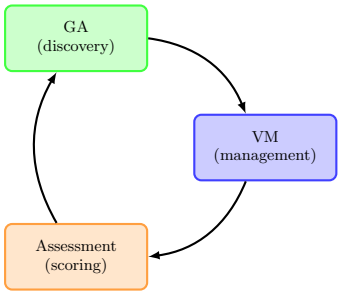


Figure 2: Three primary components of the prototype GA-based MT system and the general order of operation.

Range	CVSS score vector	Score
[15, MaxInt]	**AV:N/AC:L/AU:N/C:N/I:N/A:C**	204
(5, 15)	**AC:N/AC:L/AU:N/C:N/I:N/A:P**	213
[0, 5]	**AV:L/AC:H/AU:M/C:N/I:N/A:N**	600

Table 2: Rubric for the assignment of the CVSS score vector associated with potential vulnerabilities for Apache *KeepAliveTimeOut* configuration directive (parameter).

that can discover, implement, and assess configurations [12]. The prototype performs many of the core tasks necessary for a complete MT defense and will be used as the primary example for explaining system details in this section and Section 5. The evolution-based approach is designed to protect multiple machines.

The information flow of the evolutionary MT defense system is shown in Fig. 2, while the individual tasks are shown in Fig. 1. Three primary components interact to facilitate the discovery of a new generation of configuration parameters, to manage various hosts using the discovered settings, to assess the vulnerabilities of the configurations after a period of activation, and to pass along the qualitative configuration scores needed for the quantitative computation of fitness. A custom network protocol coordinates the three parts of the MT system, as well as the communication between the individual host machines and the system as a whole.

A genetic algorithm comprises the discovery component. The initiation of the MT system execution is signaled by the initialization of the first generation of MT chromosomes. In general, the current generation of chromosomes and associated fitness scores are passed to the VM management component. These Apache and RHEL5 configuration sets are applied to the virtual machines. After a period of time, the management component sends security information about the hosts to the assessment component. Using this information, the qualitative *bad*, *medium* and *good* scores are assigned based on the scoring rubric. These qualitative scores are passed along to the discovery component, which then computes the quantitative fitness scores for each chromosome in the current generation. The next generation of MT chromosomes are evolved using the genetic algorithm operators of selection, crossover and mutation. The next cycle of the MT system begins.

This MT defense system is proactive and resilient, since it constantly searches for new configurations. Converging to a single set of parameter settings is not the objective of the MT defense system since security will change as new vulnerabilities are introduced (e.g. via zero-day attacks or new software). The objective of the MT defense system is the discovery, management and assessment of generations of Apache and RHEL5 configurations.

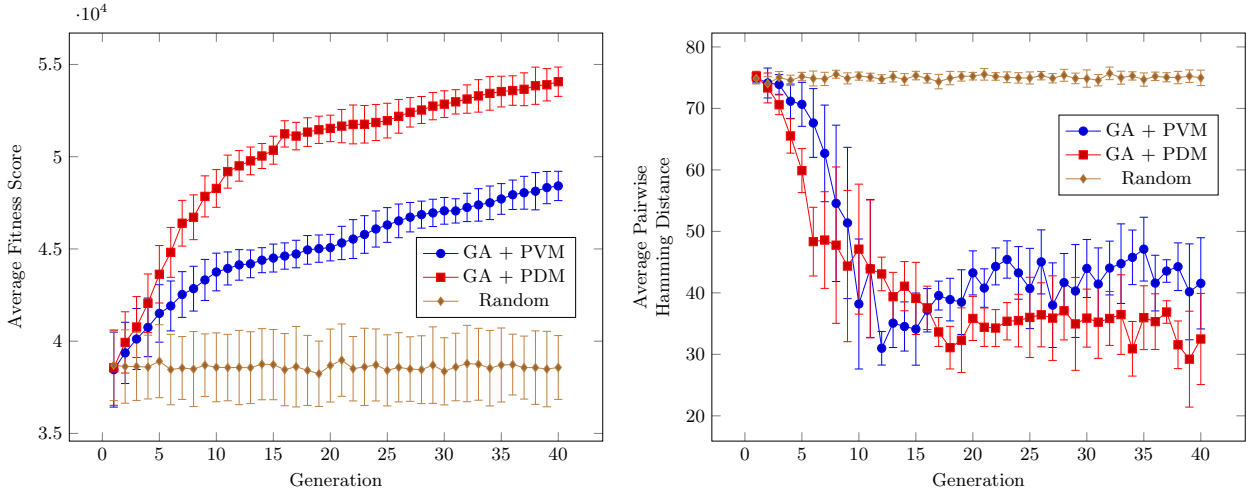
5. SYSTEM PROTOTYPE AND RESULTS

Experimental results using the prototype will demonstrate the GA-based system, starting with arbitrary initial configurations, can discover diverse and more secure configurations and create a viable MT defense for the web-server.

For these experiments, it is not practical to gather actual attacks to measure the security of the configuration. The Common Vulnerability Scoring System (CVSS) provides a method for measuring the security of an individual configuration parameter setting [13, 17], and thereby provides a method to estimate the number of attacks that will be successfully executed against a machine with the given parameter setting. CVSS scores parameter vulnerabilities; therefore, a configuration will have one CVSS score per parameter vulnerability, with the CIA scores all set to *none* to convey that the vulnerability is not present. Configuration security checklists, such as United States Government Configuration Baseline (USGCB) and DOD/NSA Security Technical Implementation Guidelines (STIGs), can then be used in combination with CVSS initially to assess the fitness of the configuration.

For this prototype system, MT chromosome fitness is the sum of values derived for the discovered parameter vulnerabilities associated with the configuration settings. Though not as realistic as gathering attack information, it does allow for a good basis for this initial simulation and measurement of attacks. The scoring rubric is specific to the parameters represented in the chromosomes and is based on expert knowledge about security parameter vulnerabilities, which can be found in NIST guidelines [16]. Each parameter is assigned a CVSS vector to represent the vulnerabilities it contains or the lack thereof. The algorithm then grades each vector's fields, assigning each a *bad*, *medium*, or *good* score based on the utility of that field's value. For example, the authentication metric can take values of *none*, *single*, or *multiple*, and these are graded as *bad*, *medium*, and *good*, respectively, as an attack requiring multiple authentications of credentials is preferable to one which requires only a single authentication, which is itself better than an attack which can be executed without providing any credentials at all. The rubric is intended to simulate the attacks faced by the computer on which the configuration is implemented. It is reasonable to assume that the estimated number of successful attacks is correlated with the severity of the security flaw as measured by the CVSS vector.

In order to quantify the fitness, the security categories of *bad*, *medium*, and *good* are then assigned numerical values of 1, 10 and 100, respectively, giving each parameter a set of six exponentially differing scores which yields a current population non-uniform probability distribution useful for selection. A chromosome's total fitness is the sum of the



(a) Average population fitness

(b) Average (sampled) diversity

Figure 3: Diversity and fitness comparisons of the genetic algorithm with parameter value mutation, the genetic algorithm with parameter domain mutation, and random search. The bars correspond to a 1 standard deviation above and below the average diversities and fitnesses, respectively. Configurations consisted of 102 parameters for an Apache 2.2 on Red Hat Enterprise Linux Revision 5.

102 individual parameter scores, and it ranges from 612 to 61, 200.

As an illustration of the assignment of the CVSS vector and the computation of the fitness score, the scoring rubric for the Apache *KeepAliveTimeOut* parameter is presented in Table 2. The *KeepAliveTimeOut* parameter takes on an unsigned integer value (measured in seconds) up to MAXINT; however, only certain ranges of values are reasonable parameter settings. The CVSS score vector reflects three ranges. From the score vector, the *KeepAliveTimeOut* parameter's fitness score is computed. This rubric reflects a more secure situation when the *KeepAliveTimeOut* value does not exceed 5 seconds.

Both genetic algorithms, GA+PVM and GA+PDM, consisted of 140 chromosomes per generation. Each chromosome represented 102 parameters. The first generation was initialized randomly and allowed to evolve for 40 generations. The 40 generations simulate the application of the MT system for over a month with daily system configuration updates. The selection operation was a blend of roulette wheel and two round tournament selection, chosen with probabilities of 0.25 and 0.75, respectively. Two point crossover, with $p_c = 0.05$, and multi-faceted mutation, $p_m = 0.02$, were implemented. One genetic algorithm employed parameter value mutation, the other used parameter domain mutation.

For comparison purposes, a random search based system was also implemented. The random search system operated similarly, except that crossover was ignored, $p_c = 0$, and mutation was always applied to every parameter of every configuration, $p_m = 1$.

Two qualitative measurements provide insight into the effectiveness of the searches. The average fitness score in a generation is one performance measure of the searches. Higher fitness scores correspond to estimates of more secure web servers. Over the generations of chromosomes, more secure configurations should be indicated by an increase in the average fitnesses. The Hamming distance between two chromosomes is the number of corresponding genes with different values, and the average of these in a generation is an indicator of the expected number of parameter differences between any two chromosomes. As the genetic algorithm moves through its generations, the average Hamming distance should decrease in value, corresponding to some agreement on secure settings. However, in order to maintain chromosome diversity, the average Hamming distance must not become too small. The computation of the average Hamming distance between all pairs of chromosomes when there are a large number of genes is computationally expensive. As an alternative, a sampling technique estimates this by choosing 15 chromosomes and computing the average Hamming distance between all pairs of this subset. In this research, this approximation has been found to be an effective measurement of population diversity.

The system based on random searching was not effective in discovering diverse and more secure configurations. Fig. 3(a) shows that the average fitness score essentially remained constant across all 40 generations. The standard deviation of fitness scores is quite wide, and does not narrow over time. Fig. 3(b) shows the average Hamming distance between pairs of chromosomes, and for the random search it remains essentially constant with a value over 74, i.e. for any pair of chromosomes the expected number of gene differences is approximately 74 out of 102. The standard deviation of the average Hamming distances is quite narrow over all 40 generations.

As GA+PVM moves through the 40 generations of chromosomes guided by the evolutionary operators, the expectation is that the average chromosome fitness and diversity should increase and decrease, respectively. The average fitness score increases 25.8% from 38, 458.9 to 48, 415.3 across the generations. As well, the fitness standard deviations suggest that almost all the chromosomes are increasing their fitness scores. Similarly, the diversity has decreased by 44.5% from 74.9 to 33.4, which indicates that the average number

of agreeing genes between pairs of Apache and RHEL5 configurations initially is 27, but in the final generation is at least 69.

For GA+PDM, similar changes in fitness and diversity of 40.2% and 56.9%, respectively, are observed. Both average Hamming distance standard deviations indicate the preservation of a diverse population of configurations. However, GA+PDM does much better than GA+PVM in terms of average fitness scores.

Increases in measured fitness correspond to more secure configurations. For any parameter, the maximum fitness score difference between a secure and insecure setting is 594 points, the difference between six partial scores of 1 and six partial scores of 100. Thus, for both GA+PVM and GA+PDM, the fitness increases of $9,956$ and $15,508$ over the 40 generations, shown in Fig. 3(a), indicate that, on average, configurations are susceptible to at least 16 and 25 fewer types of attack each compared to the initial generation.

Many parameters have settled to a subset of the values across all 140 chromosomes, and at least one of these values does not contain a security vulnerability. For instance, the *ServerTokens* parameter was set to a secure value in 15 of the final chromosomes using the GA+PVM. However, using parameter domain mutation, the domain of *ServerTokens* was reduced to only two values, and 42 of the chromosomes in the final generation are set with a secure value. This is illustrative of how later generations have become less vulnerable.

Although GAs tend to find better solutions, there is no guarantee that the final set of configurations will contain an optimal solution. This can be seen in Fig. 3(a), in which the fitness values are generally increasing, but not approaching a perfect score of $61,200$. For instance, in the first generations of the two genetic algorithms, 2 and 18 chromosomes respectively featured a *LimitExcept* setting insecure enough to warrant a score of *complete* for each of C, I, and A. In the final generations, 6 and 11 chromosomes respectively featured this score for their *LimitExcept* setting.

The *FollowSymLinks* parameter shows remarkably different settings under the two genetic algorithms. When using parameter domain mutation, the domain of *FollowSymLinks* was reduced to one setting, the secure one. For parameter value mutation, in the final generation both possible settings for *FollowSymLinks* appeared throughout the population.

6. NEXT STEPS

In the continuing development of the evolution-based MT system, there are four improvements to be added. The first is to allow the administrator to weight the CIA scores and have that weighting be reflected in the fitness function of the GA. Secondly, the management component of the MT system must be developed to detect successful attacks. Third, currently the directed mutation operator shows much promise, and with further research there is potential of significant improvement compared to the simpler mutation operation. Fourth, it is necessary to determine the functionality of a system based on a configuration.

6.1 Prioritization of CIA Scores

At the conclusion of the monitoring of the Apache web servers configured by the chromosomes, the initial CVSS scores and information about any security events are given to the assessment component. Similar to CVSS scores, the security events are scored based on the impact on confidentiality, integrity, and assurance. The administrator can weight these three components based on their relative importance. After passing this vector to the discovery component, the GA fitness function can then compute the appropriate weighted sum.

6.2 Detection of Successful Attacks

Currently, the fitness of a chromosome is determined by the simulation of attacks using rubrics developed from CVSS information. This simulation has allowed for the development and evaluation of a prototype genetic algorithm based moving target defense. However, it is necessary for the fitness of a chromosome to reflect the exposure of a computer configured with those parameter settings to the real world. It is imperative that fitness reflect security based on detected successful attacks. Ideally, the management component captures information regarding reconnaissance. Recognition of successful attacks and reconnaissance is a difficult problem.

Attack detection will be added based on checksums that can capture a limited number of successful attacks. The base fitness of a chromosome will be the value based on the CVSS rubric. This base will be adjusted based on the detection of a successful attack and the number of generations this chromosome has been implemented without a detected successful attack.

6.3 Directed Mutation

As clearly seen in Fig. 3, the genetic algorithm using parameter domain mutation does much better in terms of fitness than the one using the parameter value mutation operator. This improvement is generally reflected over all experiments. The use of simple machine learning techniques to modify the parameters' domains is an effective mechanism. However, this modification removes a value from a parameter's domain. Once the value is removed, it can not be returned to the domain. An alternative strategy is the assignment of probability distributions associated with the values of the domain. Instead of removing a value from a domain, the probability associated with that value can be decreased, allowing for increases in probabilities for other values. As well, other machine learning strategies should be considered.

6.4 Functionality Testing

An important task of the MT defense is to test that new configurations provide the necessary functionality. Regardless of how secure a configuration may be, it cannot be considered by the system if it is unable to provide the user a minimum set of required services and capabilities. This can be achieved by leveraging research that has focused on identifying and correcting incorrect configurations [11, 19]. Many of these approaches require defining predicates (tests) that can be instrumented, such as testing for a certain webpage or measuring the response delay of a system request. It would be possible to incorporate this form of testing to provide an understanding of the functionality of the configuration.

7. CONCLUSIONS

Moving Target (MT) defenses put the system to be protected in motion, in an attempt to render the reconnaissance of the attacker useless after a period of time. This article

discusses how intermittently changing system attributes directly can provide a suitable MT defense. However, finding configuration changes that maintain a functional and secure system is difficult. Configurations consist of a large number of parameters, with many of these parameters being interdependent, and may require configuration changes in concert for security to improve.

This article described how secure and diverse configurations could evolve over time based on the operational environment. Using a Genetic Algorithm (GA) and feedback about the security status of the system, this MT approach discovers configurations through a continual process of selection, crossover, and mutation. These candidate configurations can then be placed in operation over time to provide a suitable MT defense. Furthermore, the approach is resilient since it can adapt to system changes, such as the installation of new software or updates to existing applications.

Results of a prototype system show that both GAs are able to find secure configurations over the 40 generations. In addition these configurations remain diverse (configurations do differ in parameter settings), which is important for providing a MT defense. Perhaps not surprisingly, directed mutation does allow for the discovery of more secure configurations than the simpler mutation operator. Although the results are promising, more research and development are needed to reach the full potential of the system. Incorporating any feedback obtained when a configuration is made active would improve the assessment. Similarly more research is needed to measure diversity that better reflects how parameter settings are different, in terms of how they affect the system.

8. ACKNOWLEDGMENTS

The authors would like to thank Brian Lucas for his initial implementation of the GA-based MT framework as part of his 2013 Computer Science MS project at Wake Forest University.

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 1252551.

9. REFERENCES

[1] Spiros Antonatos, Periklis Akritidis, Evangelos P. Markatos, and Kotas G. Anagostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51:3471–3490, 2007.

[2] Stefan Berlik and Bernd Reusch. Foundations of directed mutation. In *Proceedings of the 2006 Conference on Integrated Intelligent Systems for Engineering Design*, pages 3471–3490, Amsterdam, The Netherlands, 2006. IOS Press.

[3] Matthew A. Bishop. *The Art and Science of Computer Security*. Addison-Wesley Longman, 2002.

[4] Crispin Cowan, Calton Pu, Dave Maier, Heather Hinton, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *In Proceedings of the 7th USENIX Security Symposium*, pages 63–78, 1998.

[5] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[6] Michael B. Crouse and Errin 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.

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

[8] Center for Internet Security. Security and assessment benchmarktools. `http://www.cisecurity.org/resources-publications/`.

[9] Stephanie Forrest, Anil Somayaji, and David H. Ackley. Building diverse computer systems. In *Proceedings of The 6th Workshop Hot Topics in Operating Systems*, 1997.

[10] Dorene Kewley, Russ Fink, John Lowry, and Mike Dean. Dynamic approaches to thwart adversary intelligence gathering. In *Proc. of the DARPA Information Survivability Conference & Exposition II (DISCEX '01)*, volume 1, pages 176–185, 2001.

[11] Emre Kycyman. Discovering correctness constraints for self-management of system configuration. In *Proceedings of the First International Conference on Autonomic Computing*, pages 28–35, Washington, DC, USA, 2004. IEEE Computer Society.

[12] Brian F. Lucas. An automated system for evolving secure systems. Technical report, Department of Computer Science, Wake Forest University, May 2013.

[13] Peter Mell, Karen Scarfone, and Sasha Romanosky. A complete guide to the common vulnerability scoring system version 2.0. `http://www.first.org/cvss/cvss-guide.pdf`, 2007.

[14] John Michalski, Carrie Price, Eric Stanton, Erik Lee, Kuan Seah Chua, Yip Heng Wong, and Chung Pheng Tan. Final report for the network security mechanisms utilizing network address translation LDRD project. SAND Rep. SAND2002-3613, Sandia National Laboratory, November 2002.

[15] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[16] NIST. Consensus security configuration checklist. `http://web.nvd.nist.gov/view/ncp/repository`.

[17] NIST. NVD common volnerability score system v2. `http://nvd.nist.gov/cvss.cfm`.

[18] Sal Stolfo, Steven M. Bellovin, and David Evans. Measuring security. *IEEE Security and Privacy*, 9:60–65, 2011.

[19] Andrew Whitaker, Richard S. Cox, and Steven 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.