Evolving Moving Target Defense Configurations

Brennan W. Gensch
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
gensc004@morris.umn.edu

ABSTRACT

Attackers of computer systems must preform reconnaissance on the system prior to attacking it. Knowledge gained through reconnaissance often allows them to find ways in which to exploit the system. Moving target defense (MTD) is a security strategy that inhibits the reconnaissance phase of an attack by making the system change the way it is configured, while it is running, at random intervals. The changes a moving target defense implements should maintain system functionality as well as security. Depending on the system being protected the possible number of parameters that can change can be very large. Finding ways to change the parameters that ensure system security is challenging, but with the help of genetic algorithms it can be done effectively. This paper describes the basics of moving target defense, highlights three specific challenges that go in to the making of an MTD, and explains how genetic algorithms can be leveraged to address those challenges.

KEYWORDS

Moving Target Defense (MTD), Genetic Algorithms, Evolution, Security

1. INTRODUCTION

A Moving Target Defense (MTD) is a security strategy implemented in a system that thwarts attackers by changing the system in ways that makes any information an attacker has gathered useless. Neutralizing the *reconnaissance* phase of an attack can stop the entire attack [2].

There are three challenges that an MTD faces before it can be a successful defense strategy: *unpredictability*, *coverage*, and *timeliness* [6]. Genetic algorithms can be used to address these challenges, and assist in the creation of a powerful MTD.

This paper will discuss the basics of moving target defenses, outline the challenges an MTD faces, and show how genetic algorithms can be used effectively to create a moving target defense.

UMM CSci Senior Seminar Conference, May 2016 Morris, MN.

Moving target defense and the basics of genetic algorithms will be defined in the background section, and how genetic algorithms can be leveraged in the creation of a moving target defense will be described in sections 3 and 4.

2. BACKGROUND

2.1 Moving Target Defense

The idea of a MTD is to take a static system and make it *dynamic*. A system can be defined as a computer or server that needs to be protected, and a static system is one whose characteristics or properties are not changed once deployed. To make a system dynamic the *configuration*, or set of system properties, must be changed. This change should occur at some frequency, potentially random, and the configurations implemented by the MTD must maintain system functionality as well as protect the system without disrupting its services.

2.1.1 Purpose

Systems whose properties remain static are in danger of being exploited. Their static nature allows attackers of the system to study the configuration to find potential vulnerabilities and exploit them. Studies show that attackers spend around 70 percent of their time on this research, or *reconnaissance* [1]. Making a system dynamic helps limit the usefulness of reconnaissance, because potential vulnerabilities that exist in one configuration may be non-existent in another that is used later in the system's lifetime.

2.1.2 Creation

MTDs can protect a system by targeting three different levels of the system. These levels are: *memory*, *network configurations*, and *host-level infrastructure* [1].

An MTD that focuses on making the storage of memory dynamic would involve changing where parts of memory are stored in the system. This change would help deter attacks like *buffer overflow attacks* because attackers could not make assumptions about where interesting information is located. Many systems already implement *address randomization*, which can be considered a form of moving target defense. Address randomization alters the location of data and instructions a program is using in memory throughout its run-time. This method is effective for protecting private information, but is only applicable to single-application systems[2].

An MTD operating in the network level of a system aims to modify the network addresses of parts of the system. At-

tackers of a system that implemented a dynamic network configuration would be unsure of what computer they were interacting with at a given time. This particular form requires the collaborative use of many resources, and can be difficult to implement. Distributed systems, like a computer lab, may need to have each part the system know how to find the other parts of the system. If an MTD is changing the way a system can be found, then all of the systems need to know that the change was made and when it was made so that the system as a whole can maintain its functionality.

An MTD created for host-level infrastructure focuses on changing multiple system properties directly[1]. Genetic Algorithms are most useful in the creation of an MTD that affects the host-level infrastructure, and this area will be the focus of the rest of the paper.

2.1.3 Implementation

Before a moving target defense can be implemented, a set of configurations must be created that the moving target defense can use to change the system it is protecting. The set of configurations being used must all be secure and maintain normal system functionality once implemented. These configurations can be found by implementing a genetic algorithm. The creation of configurations will be discussed more in section 3.

To ensure that configurations are safe they are tested first. Testing a configuration can be done in different ways, but the most common way is to emulate the configuration on a virtual machine and run tests to determine its security.

The next step is to switch the configuration of the production system without interrupting its services. The frequency at which this change happens is different depending on the MTD, but the goal is to make the change between when an attacker has done their reconnaissance and when they try to start an attack.

2.1.4 Challenges

There are three main challenges in the creation and implementation of an MTD. The challenges are accomplishing unpredictability, coverage, and timeliness[6].

For a system to achieve unpredictability an attacker must not be able to predict future movements, or changes, to the system. If the attacker is able to obtain information about potential configurations that will exist on the system in the future, then they would be able to plan on attacking that particular configuration. If the dynamic system is predictable, it is essentially static to a competent attacker.

The coverage of an MTD refers to its ability to impact all of what is called the *attack surface*. The attack surface is the area of a system that an attacker may use to negatively impact the system. An MTD with perfect coverage would make 100% of the attack surface dynamic. If parts of an attack surface are left static then it is possible for the system to be exploitable no matter how dynamic the rest of the attack surface is. To find what level of coverage is needed a *threat model* must be defined for the system. A threat model is a description of threats that exist for a particular system, and must be created in the context of that system [6].

Timeliness is the most difficult property to achieve in an MTD. The timeliness of a system is in reference to the frequency in which changes are made to the attack surface. The goal of an MTD is to make changes to the system between an attacker's reconnaissance and their attack. Changes to the system must be frequent enough so that attackers do not have a lot of time to learn about the system, but it should not be so fast that it impacts the performance of the system's normal functionality. It is also important for the frequency to not be predictable [6].

2.2 Genetic Algorithms

2.2.1 Evolutionary Concepts

Evolutionary computation is a field of artificial intelligence that attempts to replicate the process of biological evolution to find solutions to complex problems by evolving the solutions [4].

Evolutionary computation is useful when applied to problems that cannot be computed efficiently in polynomial time. Problems that can only be solved by checking every possible answer are computationally difficult to solve. Evolutionary computation does not focus on solving problems algorithmically or mathematically, but finding solutions by starting with a group of *candidate solutions* and evolving them to be better. Candidate solutions are generally created by choosing random answers the the question that is trying to be solved. A common process of evolution looks like this: take a random solution in a set of possible solutions to a problem, create a *generation* of solutions that are modifications of the original, and use the best of the new generation to create the next.

2.2.2 Genetic Concepts

Genetic algorithms are a form of evolutionary programming that is used to find solutions to search problems [5]. Genetic algorithms start by initializing a generation of candidate solutions, often called chromosomes, that have sets of modifiable properties. The algorithm takes that generation and performs three steps: *selection*, *crossover*, and *mutation*.

The selection process looks through the current generation of chromosomes, or candidate solutions, and determines which will be used to create the next generation. Chromosomes are selected in different ways depending on what selection function is used, but the choice is always dependent on chromosome's *fitness score*. A chromosome's fitness is a measurement of how close the individual is to solving the problem. The fitness score is different depending on the problem, and must be determined before a genetic algorithm can be created.

The crossover function is used after two or more chromosomes have been selected from the generation. Two chromosomes are taken, and some of their properties are randomly swapped. This process is meant to emulate the crossing over of genes between parents to create children. After the properties are swapped these two new chromosomes are used to create the next generation.

The mutation process takes the two or more chromosomes that the crossover function was applied to and creates new chromosomes by editing some of their properties at random. This process is repeated to generate chromosomes for a new generation. The process of selection, crossover, and mutation is repeated until chromosomes that are suitable answers are generated.

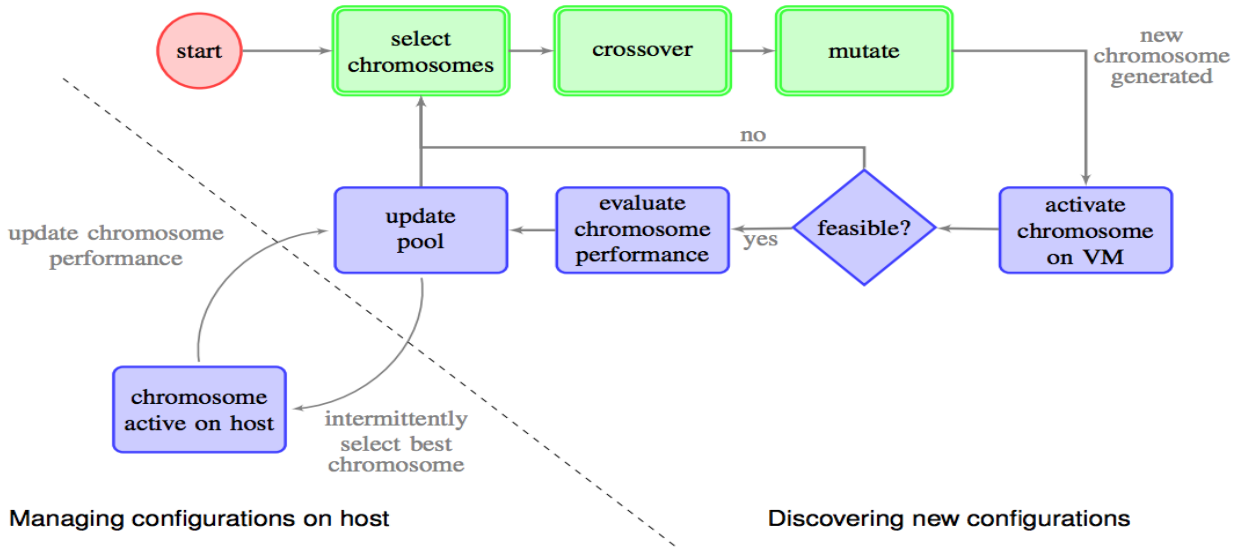


Figure 1: Process of evolving the configurations for an MTD

3. LEVERAGING GENETIC ALGORITHMS IN MTD

As mentioned in section 2, genetic algorithms are most useful in a host-level infrastructure MTD. Table 2 is an example of what properties would exist in a configuration targeting the host-level infrastructure an Apache web server system [1]. There are a variety of values that these properties can have, and these values determine how the system functions. System properties like those in table 2 are what attackers look at when performing reconnaissance on a system; thus, these are the properties that will be changed by an MTD.

In an Apache web server the *keepAlive* parameter tells the system if it should accept multiple requests from a single connection. This knowledge is something an attacker could take into account when trying to find vulnerabilities in a system.

An MTD needs a large number of secure configurations that maintain functionality to operate effectively. The number of configurations that could exist is dependent on the size of parameters in the system's configuration. As the number increases, it becomes difficult to search for good configurations [1]. Genetic Algorithms can search through the space effectively, and discover secure solutions.

Figure 1 shows how an MTD operates when using a genetic algorithm to create configurations. Configurations, or chromosomes, go through the genetic operations and are then run on a virtual machine. Once on the virtual machine, their feasibility is determined. A feasible configuration will have attained a certain level of fitness. The measurement of fitness will be defined in section 3.1.2. If a feasible configuration is found, it is added to a pool of configurations that will be used on the real system at some point in the future.

Parameter	Value Type
`.htpasswd`	Binary
`ServerTokens`	Value from a list
`KeepAlive`	Binary
`KeepAliveTimeout`	Positive integer
`FollowSymLinks`	Binary
`IncludesNoExec`	Binary
`Indexes`	Binary
`LimitRequestBody`	Positive integer

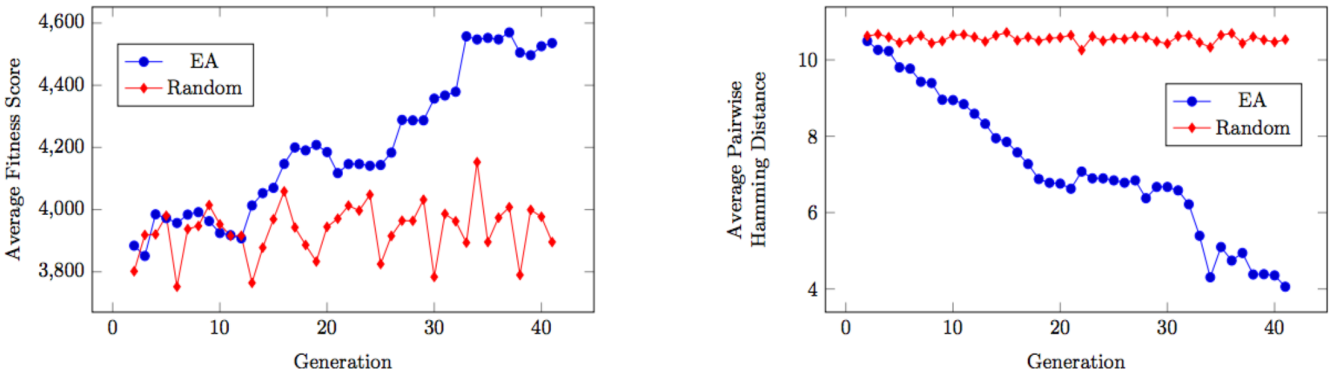
Figure 2: Example of parameters in an Apache system [1]

3.1 Evolving Configurations

Genetic algorithms are used to evolve better solutions to a variety of challenging problems. This section will discuss specific methods used to create a genetic algorithm (GA) that will be effective for evolving MTD configurations.

3.1.1 Scoring: Measuring Fitness

In this problem space, the fitness of a single configuration represents the security of that configuration. This security is determined by assessing the affects a configuration has on a systems confidentiality, integrity, and assurance. The confidentiality of a system refers to the protection of information from those who should not have access to it. Integrity is defined as the ability of a system to contain consistent and correct information. Assurance is the availability of information and system functionality. Each parameter of the configuration is scored on how it impacts these aspects of the system, and the combination of these scores creates the configurations fitness score [1]. In some MTD systems, the fitness score is generated by putting the configuration on a virtual machine and running a set of tests on it. This step is shown in figure 1 as activating the chromosome on the VM.



(a) Average population fitness.

(b) Average (sampled) diversity.

Figure 3: Results of implementing a GA on 14 Apache web server parameters [1]

3.1.2 Selection, Crossover, and Mutation

Selection

Tournament Selection is a selection function that randomly selects a small number of configurations and compares them to one another. The best of that tournament is used as a parent to create the next generation. Multiple tournaments are performed until the desired number of parents are selected. This number is normally a small subset of the original generation, and the goal is to find the best configurations while maintaining some amount of *diversity* in the parents. Diversity is an important aspect in genetic algorithms, and particularly import in reference to the creation of an MTD. Diversity in a set of configurations can be defined as how the variation between configurations. Having good diversity in the set of parents helps ensure that the algorithm as a whole is exploring enough of the space. Exploring the space of the problem is important to a point; however, if too much exploration is done the algorithm can become useless as it just finds random solutions at every level of the evolutionary process instead of actually evolving the solutions it has found.

Crossover

After selecting the parents, a crossover function is used on two parents to create two new children. An example of a crossover function that could be implemented is called uniform crossover. Uniform crossover takes a number of random properties of each configuration and swaps them. Doing this would prove useful because property values that have resulted in good configurations have the chance of being combined with other good properties from other configurations. This will result in making configurations that are filled with combinations of different secure properties, which is what an MTD needs to thrive.

Mutation

After children are created from the crossover function they are mutated. There are not any specific mutation functions that are useful in an MTD, but the process of randomly changing a small number of properties is important for the diversity of the generations created in the future [5].

Repeating the process of selection, crossover, and mutation will eventually result in a generation that contains configurations that are sufficiently fit and secure. Once configurations are found and rated for fitness, a small selection of the best will be tested by the system. If they are fit enough to be used by the production system they will be put into a pool of potential configurations that the MTD can use, and the evolution process will continue. The pool of configurations has a specific number of configurations it can contain, determined by whoever is implementing the system, and if configurations with higher fitness are discovered then the worst of the pool will be removed to make room. At some undisclosed time the MTD will replace the current configuration on the host with a configuration from the pool. This whole process is shown in figure 1 [1].

3.2 Results of implementing a GA

Figure 3 shows the average population fitness and average sampled diversity of a genetic algorithm implemented by Lucas et al [2], which evolved configurations that consisted of 14 Apache web server parameters. The blue line shows the results of the algorithm, and the red line shows the results of creating random configurations for each generation. Part A shows that the genetic algorithm succeeded in creating more fit, more secure, configurations than was possible with making random configurations. The configurations created by Lucas et al [2] succeeded in the creation of usable configurations for an MTD.

3.3 Addressing MTD Challenges

Using genetic algorithms in the way described above effectively reduces the predictability of a moving target system. A genetic algorithm results in a set of configurations that have been mutated from a child that was created by randomly crossing two parents from the previous generation. The configurations that exist in the final generation are not predictable. Genetic algorithms tend to find solutions that are strange in some way due to the randomness the system creates. The solutions that are found will be secure and maintain system functionality, but will not be what an attacker expects a normal configuration to look like.

Another benefit of using a moving target defense that leverages genetic algorithms is that the evolutionary process

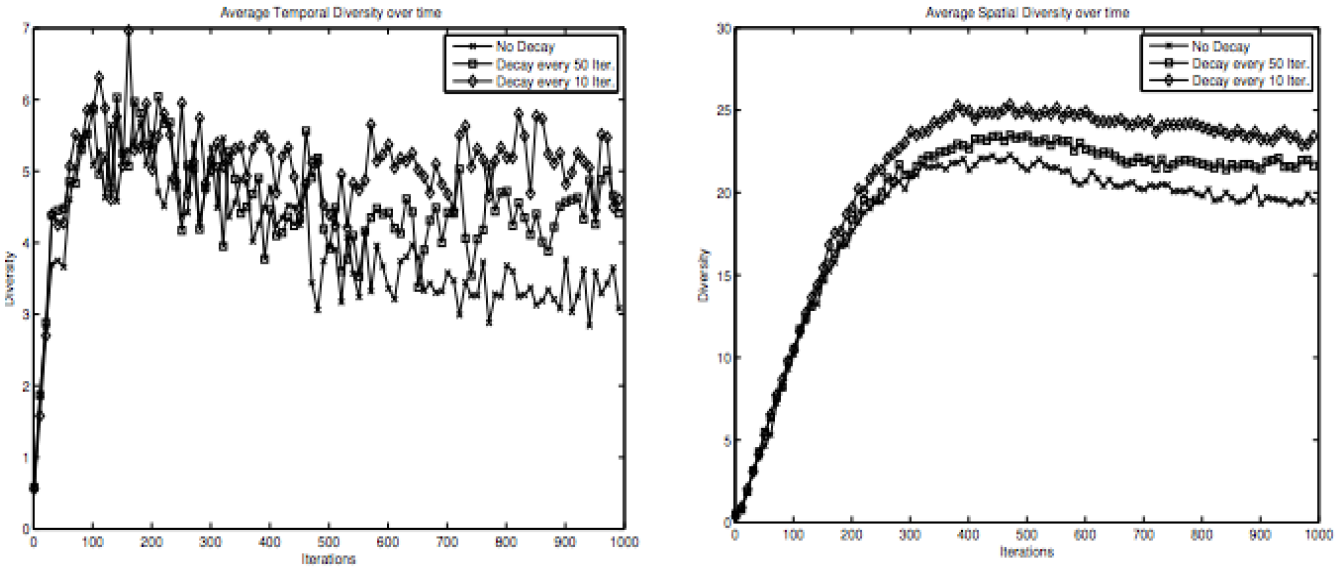


Figure 4: Results of implementing a fitness decay algorithm [3]

does not need to stop once the system being protected is live. This means that configurations do not need to be reused or stored for long periods of time as new configurations are always being created. Attackers of this kind of system will not know what to expect, nor will they be able to find the configurations before they are implemented. Consequently, genetic algorithms help remedy the problem of predictability in MTD, and increase security through obscurity.

Using common genetic algorithms helps combat the challenge of predictability, but the genetic algorithm must be implemented in a way that ensures diversity to stay unpredictable as well as address other challenges [6]. In Figure 3 part b the average sample diversity is shown. Again, the red line represents random creation of configurations and the blue line represents the genetic algorithm. Diversity was calculated with what is referred to as the *hamming distance*. Hamming distance is calculated by comparing two sets of values that are the same length. Each differing value increases the hamming distance by one; therefore, the hamming distance of configurations shows how many parameters in the configuration are different from the configuration it is being compared against. The diversity of the configurations decreases significantly throughout the generations of the evolutionary process, which could negatively impact the usefulness of the MTD, specifically in regards to its coverage.

4. INCREASING DIVERSITY

As shown in the above section, it is possible for generations to converge on a small subset of configurations. If the generations lose their diversity less unique configurations are made, and it becomes easier for an attacker to predict what configurations may be used in the future. Another weakness of less diverse generation is that the coverage of the MTD can be reduced. If configurations are too similar, fewer properties are actually being made dynamic when the configuration of the host is changed. One way to remedy this problem is to increase diversity by implementing what is called a fitness decay algorithm [3].

4.1 Fitness Decay Algorithm

In order to increase diversity Crouse et al [3] implemented a fitness decay algorithm that systematically reduces the fitness of chromosomes, or configurations, in the pool of configurations being used by the MTD. The reduction in fitness over iterations of the MTD cycle allows for configurations that are unique to be added to the pool. This will stop the pool of potential configurations from stagnating.

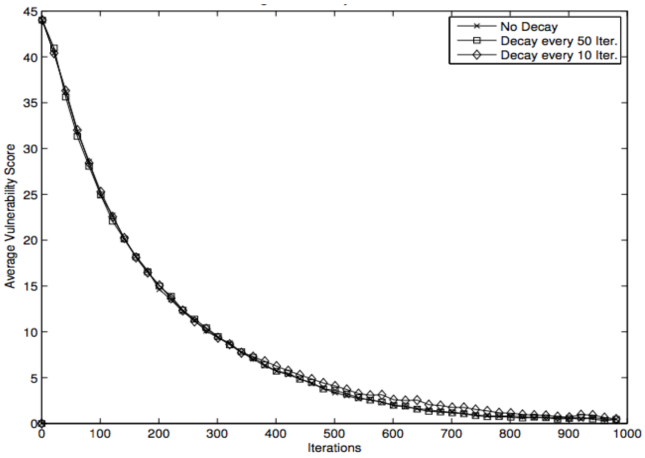


Figure 5: Average vulnerability score of the pool over time [3]

4.1.1 *Results of implementing the fitness decay algorithm*

Figure 4 shows what happens to the diversity of the system when a fitness decay algorithm is included in the genetic algorithm. The graph on the left describes the *temporal di-*

versity with three different decay strategies while the right shows average *spatial diversity*. This particular study was performed on a distributed system containing 60 different computers of the same type. Temporal diversity is measured by the hamming distance of the new configuration compared against the last configuration that was running on the machine. The spatial diversity is measured by the average hamming distance between all 60 configurations that are running on the machines during one cycle of the MTD [6]. The best of both graphs is achieved when configurations have their fitness decayed every ten iterations. This shows that the diversity, or hamming distance, is the highest when configuration's fitness is decayed every 10 generations. Decreasing the fitness of configurations in the pool will allow for new configurations that have not been used by the system before to enter the pool and be used by the host.

Figure 5 shows the average vulnerability score of the configurations in the same pool shown in figure 4 after implementing the fitness decay algorithm. The vulnerability score in this particular study is the inverse of the fitness score described in section 3. The smaller the vulnerability score a configuration has the more secure it is. This graph shows that even though we decay the fitness of configurations in the pool the average vulnerability score does not differ from not decaying the fitness. This shows that using the fitness decay algorithm properly increases the diversity of the system while also maintaining system security and functionality.

4.2 Addressing MTD Challenges

Since the fitness decay algorithm ensures diversity within generations, the final generation should include configurations that are secure and explore the space sufficiently enough that the system's attack surface is covered. The average hamming distance represents the average number of differences in parameters between chromosomes in the pool. This implies that if the hamming distance is larger, then a larger number of parameters will be changed when the MTD changes what configuration is on the host. A larger number of changes between each change will ensure that the attack surface the parameters impact will be properly covered. With the attack surface covered by the possible configurations it will be challenging for attackers to find static components they can exploit.

Unpredictability can be addressed using normal genetic algorithms, and coverage can be addressed by using a fitness decay algorithm. Timeliness is a hard problem to address, and has yet to be solved by researchers in the moving target field. Knowing when to change a systems configuration is dependent not only on the system, but also on the type of attack and the skill of the attacker. Timeliness may not be solved currently by the use of genetic algorithms, but future research may find interesting ways to make an MTD even better.

5. CONCLUSION

Moving target defense creates security in a system by making it change itself throughout time. Attackers of a system that implements an MTD will not know what they are attacking, and the time they spend trying to research and understand their target will be wasted. Genetic algorithms can create configurations that address the challenges of unpredictability and coverage, and the challenge of timeliness is an area of future work for researchers in the field. The genetic algorithm described in this paper, created by Lucas et al [2], is a simple genetic algorithm, but it is effective in the creation of secure configurations that can be used by an MTD. The fitness decay algorithm implemented by Crouse et al [3] succeeded in increasing the diversity of configurations created by genetic algorithms in a way that increases the average coverage of the system. Moving target defenses protect systems not by decreasing their vulnerabilities, but by making those vulnerabilities harder to find and exploit.

6. ACKNOWLEDGMENTS

I would like to thank K.K. Lamberty for advising me throughout the process of writing this paper, and I would like to thank Elena Machkasova and Alex Jarvis for their corrections on my drafts.

7. REFERENCES

[1] D. J. John, R. W. Smith, W. H. Turkett, D. A. Cañas, and E. W. Fulp. Evolutionary based moving target cyber defense. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1261–1268, New York, NY, USA, 2014. ACM.

[2] B. Lucas, E. W. Fulp, D. J. John, and D. Cañas. An initial framework for evolving computer configurations as a moving target defense. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, CISR '14, pages 69–72, New York, NY, USA, 2014. ACM.

[3] E. W. F. Michael B. Crouse and D. Cañas. Improving the diversity defense of genetic algorithm-based moving target approaches. In *Proceedings of the National Symposium on Moving Target Research*, MTD '12, 2012.

[4] Wikipedia. Evolutionary computation, 2016. [Online; accessed 19-March-20016].

[5] Wikipedia. Genetic algorithms, 2016. [Online; accessed 19-March-20016].

[6] R. Zhuang, A. G. Bardas, S. A. DeLoach, and X. Ou. A theory of cyber attacks: A step towards analyzing mtd systems. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 11–20, New York, NY, USA, 2015. ACM.