Dynamic Application Rotation Environment for Moving Target Defense

Michael Thompson
Global Security Sciences
Argonne National Laboratory
Argonne, IL, USA
thompsonm@anl.gov

Michael Muggler
Department of Computer Science
University of Texas at Dallas
Dallas, TX, USA
mxm121531@utdallas.edu

Marilyne Mendolla
Department of Computer Science
University of Texas at Dallas
Dallas, TX, USA
mxm122230@utdallas.edu

Moses Ike
Department of Computer Science
University of Texas at Dallas
Dallas, TX, USA
mji120030@utdallas.edu

***Abstract*—Owing to the ubiquity of web applications in modern computing, the server software that delivers these applications is an attractive attack vector for would-be malicious actors in cyberspace. Recently, Moving Target Defense (MTD) strategies have grown in popularity in the computer security community because of their ability to enhance resilience and force attackers into uncharacteristic behavior. The MTD prototype discussed in this paper acts as a proactive defense strategy that offers increased protection against an attacker's ability to probe for and exploit vulnerable web server software. The testing shows that web server diversity in an MTD reduces the ability to exploit vulnerabilities in a web server, reduces impacts of successfully exploited vulnerabilities, and increases the resilience of the protected application.**

Keywords—self-adaptive technologies; resilience computer and control systems; robust systems; cybersecurity; moving target defense; computer security; computer hacking; Internet; cyberspace; security; proactive defense; operating systems; platform diversity

I. INTRODUCTION

Cyber-attacks pose a major threat to critical infrastructure [1]. The static nature of defense mechanisms and application infrastructure makes it easier for an attacker to fingerprint and identify possible attack vectors [2]. While strategies and defensive techniques are many and diverse, the Moving Target Defense (MTD) technique has recently gained attention as a possible solution for mitigating cyber-attacks [3]. Argonne National Laboratory has developed the Multiple Operating System Rotational Environment MTD (MORE MTD) solution, which employs a strategy of combining the use of operating system diversity with a dynamically changing footprint to increase defenses [4]. A simple extension of MORE MTD is to take the aforementioned strategy and apply it to the web server platform. In this paper, we introduce a Dynamic Application Rotation Environment for Moving Target Defense (DARE MTD) as a strategy to extend the ideas proven in MORE MTD to other parts of the attack surface. Though the current research data focuses on the effectiveness of DARE MTD alone, future work will explore a layered approach that combines both strategies.

DARE MTD uses the two most common and freely available web servers, Apache and Nginx. It runs a single application on both platforms, redirecting incoming traffic to one server or the other at a random interval. The goal is to mitigate any unknown vulnerability in one of these platforms by reducing the amount of time that platform is exposed to a would-be attacker. Like the MORE MTD strategy, this variability increases the cost of reconnaissance on a target and reduces the likelihood of exploiting any zero-day, or previously unknown, vulnerability.

II. BACKGROUND

MTD techniques are categorized into shuffle, diversity and redundancy [3]. The shuffle technique rearranges the configuration of operating systems or applications. The diversity technique randomizes the operating system or software stack components. The redundancy technique requires multiple replicas of network components where paths through the network are randomized. These techniques can be applied either proactively or reactively.

A proactive MTD anticipates adversarial behaviors by continuously changing in some randomized fashion, while a reactive MTD adapts as a preventative measure when an attack occurs [3]. As established in the research underlying MORE MTD, a proactive approach to cyber defense maintains the integrity and accessibility of the application(s) in use [4]. Work done by Thompson et al. [4] implements a proactive MTD by rotating the operating system on which a web application runs.

978-1-5090-2002-7/16/$31.00 ©2016 IEEE

In this setup, the same application runs on several virtual machines (VMs) with different operating systems.

One VM is selected at a given time to handle all network traffic, and it is known as the active VM. At a predefined interval, which may be as short as 15 to 30 seconds, the active VM is switched. When a VM becomes inactive, the integrity of the file system is checked for signs of attack and removed from rotation if any integrity compromise is detected. The procedure mirrors MORE MTD, which set the lower rotation window to 60 seconds [4]. The idea of both strategies is to rotate at a suitable interval in order to accomplish the following:

1. Prevent accurate fingerprinting and identification of entry points.
2. Thwart persistent attacks by reducing the exposure of vulnerable software.
3. Reduce the viability of any gain to the attacker by consuming extra time and resources.

Existing MTD studies do not address a formal security model in which to assess the benefits of MTD. The current state-of-the-art approach for evaluating the security effectiveness is Attack Representation Models (ARMs) such as Attack Graphs and Attack Trees [6]. Hong and Kim prefer a Hierarchical ARM, which is both scalable and adaptable, over other ARMs. However, these approaches consist of performing structured logical arguments rather than either conforming to formal models or building and testing real implementations.

It is easy to add defenses that provide few security benefits while introducing unacceptable overhead, inadvertently increasing the attack surface or causing unintended and unforeseen side effects. In the paper by Soule et al., Attack Surface Reasoning (ASR) is introduced as a method to quantify the security, performance, and overhead of MTD implementations [14]. ASR focuses on understanding the cyber systems in a well-defined manner, considers feasible attack vectors, and compares the tradeoffs between increased security and costs across multiple configurations.

III. System Design Goals

Our goals in designing DARE MTD are centered on the two themes of increasing attacker uncertainty and increasing defensive system resilience. To measure these properties, we established three primary measurable goals, as inspired by those of MORE MTD:

1. Reduce the likelihood of a successful exploit.
2. Reduce the impact of a successful exploit.
3. Maintain application availability.

These three goals map across our two themes, as shown in Fig. 1.

We verified the success of these goals in two primary ways: (1) web server and VM fingerprinting, and (2) the ability to mitigate known vulnerabilities in web server software. Specific details about the laboratory and real-world testing scenarios and results are provided in the following sections.

A. Increasing Uncertainty

In attempting to increase attacker uncertainty, we want to eliminate the attacker's advantage in having virtually unlimited

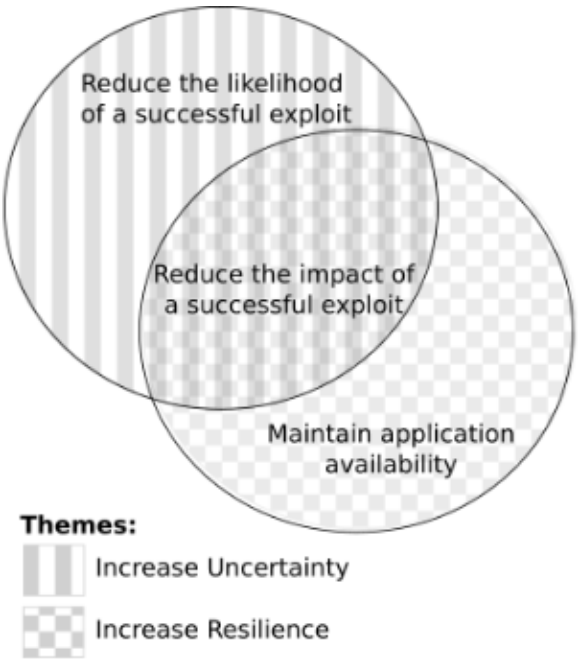

Fig. 1. Goals and themes.

time to gather information about a system [10]. A normal attack involves some level of reconnaissance to determine the vulnerabilities of a system. By constantly shifting our vulnerability footprint, we eliminate the value of any intelligence gathered by a would-be attacker, as proven by MORE MTD research [4]. The addition of random rotation orders starts with this foundation and increases the uncertainty even more.

B. Increasing Resilience

Resilience within computer systems is generally taken to mean dependability and performance in the face of unanticipated changes [11]. In a cyber-system or a cyber-physical system, our goal is to increase robustness while reducing brittleness. Fig. 2 shows idealized response curves of both resilient and un-resilient systems to demonstrate these properties of robustness and brittleness. The three curves in the graph represent three scenarios of impact to a system. The resilience threshold represents the maximum amount of performance loss an application or service can take and still be usable. Agility represents the time/ability for the application to return to ideal operating characteristics after the onset of a negatively impactful event. Brittleness is the measure how long an application/service stays unusable after it becomes unusable.

DARE MTD increases an application's resilience threshold, as described by Rieger [13], by reducing the rotation window. Though this reduction does not affect the brittleness of an individual web server, the application grows in overall resilience by residing on multiple platforms. We improve resilience through rotation of web servers with different

vulnerability footprints. We improve agility and robustness through increasing uncertainty and reacting to failure. For the case of a security system, this means both attempting to prevent unanticipated execution of code and recovering sanely from any such unanticipated execution. As with MORE MTD, our primary goals are to reduce the likelihood of a successful exploit, reduce the impact of any successful exploits, and keep the application up and available both during rotation and in the event of a successful exploit.

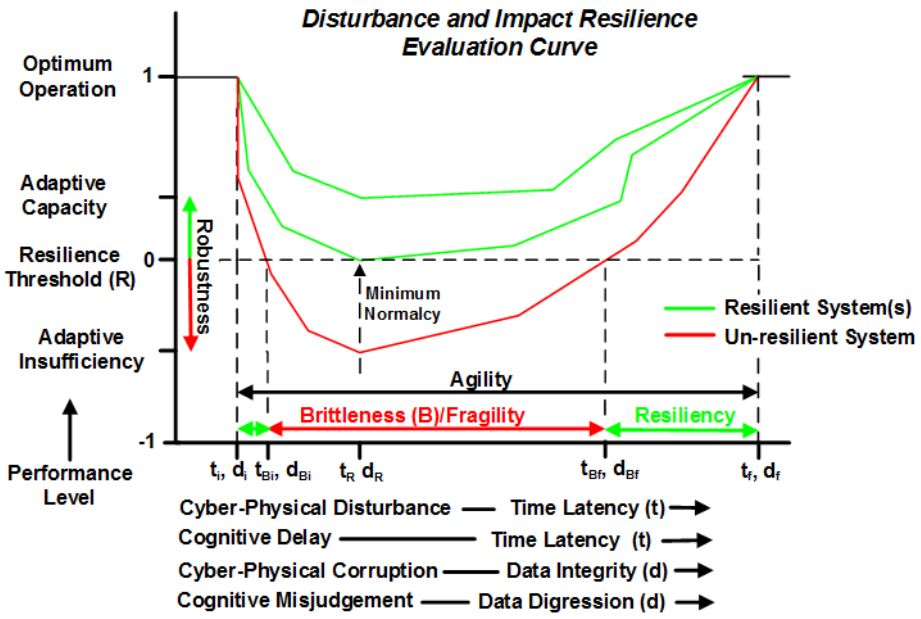

Fig. 2. Cyber-physical resilience curve. [13]

IV. DARE MTD Implementation

An essential aspect of MORE is that it utilizes already-existing technologies for MTD implementation [4]. Thus, the DARE MTD prototype discussed here was deployed in as realistic an environment as possible. To accomplish this goal, we used common software and configurations from a typical production environment and tested using common vulnerability scanners, penetration testing tools, and frameworks. Testbed components can be seen in detail in Table 1. The DARE MTD Mechanism rotates between two actively running web servers on a single host, both supporting a single, stateless web application.

TABLE I.

VM	Software	Purpose
MTD VM	CentOS	Operating System
	Apache HTTPD Server	Web Server #1. Port 82
	Nginx Server	Web Server #2. Port 81
	WordPress	Stateless Web Application
	Web Server Rotation	**Custom Service That Implements MTD Using IP Tables**
Reconnaissance VM	Kali OS	Operating System
	Nessus/Nmap/OpenVAS	Fingerprinting
	Metasploit Framework	Penetration Testing

For testing purposes, DARE MTD and penetration testing software were deployed on separate VMs contained on an individual host machine. This allowed for a consistent networking environment and a versatile template that could later be deployed to other testbeds and production environments. Fig. 3 describes the networking environment as it existed for the tests outlined below.

DARE MTD takes incoming web traffic (on standard TCP port 80) and redirects it to one of two ports: (1) nonstandard TCP port 81 for traffic served by Nginx or (2) nonstandard TCP port 82 for traffic served by Apache. The MTD mechanics are as follows:

- Host-based firewall configuration: Linux IP Tables is used to redirect incoming web traffic to either the Nginx port or the Apache HTTPD port.
- Web servers: both Nginx and Apache HTTPD are configured to listen to localhost on their respective nonstandard TCP ports.
- Rotation service: a custom script (running as a service), which performs the IP Tables rule, updates to redirect the web traffic from TCP port 80 to either the Nginx or the Apache HTTPD TCP port. This update occurs at a random yet bounded time interval.

A web server in a production environment typically listens for incoming traffic on port 80 (HTTP). As such, our MTD mechanism can be installed on typical Linux production servers with minimal effort and thus adheres to the MORE MTD ideal. The web servers are configured to listen to localhost, which makes them unreachable on the network. IP

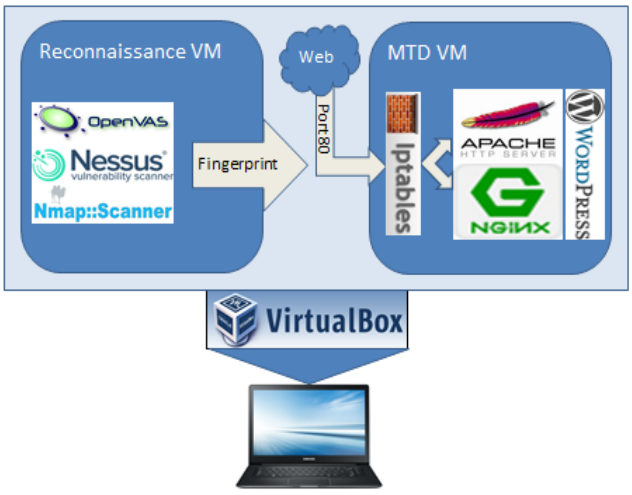

Fig. 3. Virtualized Prototype Implementation of DARE MTD

Tables are configured in a way which only allows traffic on the TCP ports that have been defined. In our design, we have made no assumptions about the enterprise configuration of a system, and therefore do not rely on network firewalls or other mechanisms to aid in the MTD.

V. Laboratory Tests

A. Moving Target Defense Virtual Machine Fingerprinting

One piece of our MTD analysis is to determine if the Web Server MTD mechanism prevents accurate fingerprinting of the MTD host. Preventing an accurate fingerprint is important, as it delays or even thwarts attacks. To analyze this aspect, we used several popular and freely available fingerprinting tools.

Nmap is an open-source tool that is used to discover hosts and services running on a computer network, and it was also utilized during the MORE MTD research [4]. For our analysis, we ran the Nmap application against our MTD VM, scanning only port 80. As expected, multiple scans reported different results—either the Apache HTTPD server or the Nginx server was reported.

Nessus is a vulnerability scanner developed and maintained by Tenable Network Security. It is free of cost for personal use in a non-enterprise environment. We ran a custom scan against the MTD host, focusing on the Web Server family of plugins only, to detect only the vulnerabilities in the web servers running on the host.

Baseline Nessus scans were run against the Nginx server and the Apache HTTPD server without the MTD mechanism in place. Once the results were verified, scan results were conducted against the host that ran the MTD mechanism. The first scan found that the Nginx server was the HTTP server that ran during the scan. Interestingly, however, a Medium finding was detected during this scan (no Medium finding was detected during the baseline Nginx scan), and that finding reported an HTTP trace vulnerability. Nessus sent a trace packet to the server, and the server responded that Apache HTTPD was running. In a subsequent scan, the Apache HTTPD server was detected, but no Medium vulnerability was detected (there was one Medium finding in the Apache HTTPD baseline scan, however).

The MTD scan results show that fingerprinting an MTD host is difficult because the scans report conflicting or inaccurate findings. This makes planning and executing an attack more difficult for a potential attacker -- overall risk is reduced by increasing the difficulty for bad actors to plan and execute an attack.

B. Moving Target Defense Exploit Mitigation

CVE-2011-3196 describes a vulnerability in Apache Web Server 2.2.15 [5]. According to the CVE information, the byte range filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remote attackers to cause a denial of service or DOS (memory and CPU consumption) via a Range header that expresses multiple overlapping ranges, as exploited in the wild in August 2011. The CVE-2011-3196 vulnerability has a very high CVE score of 7.8. We hereby exploit this vulnerability within our MTD architecture and show that MTD mitigates the exploit.

1) Attack Methodology

This vulnerability can be exploited through a request and payload-string to Apache, which forces it to perform a very memory and cpu-intensive operation. When multiple request/payload strings are sent to the Apache server, Apache processes begin and become extremely stagnant (not operational/exhibiting hangs), extremely slow, or killed because of resource request timeouts. This process results in Apache's inability to service any more requests because it has depleted the available resource (which is needed to service incoming requests). In addition, because it is a DOS attack, the attack takes some time (8-12 seconds in our case) to render Apache out-of-service. Hence, DARE MTD extends the MORE MTD idea of minimizing the rotation window.

2) Moving Target Defense Effectiveness

This exploit becomes ineffective, when the rotation script switches applications, because Nginx is not vulnerable to this particular attack or request-payload. Therefore, Nginx does not perform any resource-intensive operation like Apache but instead simply replies to the attacker as for a normal HTTP request. The period when Nginx is active allows Apache to recover from the ongoing DOS attack. Apache simply drops all its connections, since it now sees the connection as orphaned (the requesting client is no longer there). As soon as Apache becomes the active server, though, the DOS exploit begins once again to build up until MTD switches to Nginx.

Suppose an integrity/attack check is performed on the rotated web server once it becomes passive. It will be evaluated, the attack will be detected, and necessary action will be taken. In conjunction with the MORE MTD study, the implication is that though an attack may occur during a given rotation window, the damage may be mitigated through rotation and incursion checks [4].

3) Testing Procedure

1. Before testing, limit the amount of memory and CPU available to your VM. Otherwise, Apache might use 100% of your machine resources.

2. Without MTD, run the exploit against the Apache server and confirm that you can no longer acquire service from the server. Note the time required for Apache to become out of service for your case. Run the TOP command to monitor your process's resource consumption.
3. Stop and prepare MTD.
4. With MTD, run the exploit. Run the TOP command to monitor your progress. In addition, gather additional information about the optimum rotation interval and other observations.

4) Mitigation Results/Observation

As per our rotation configuration, the maximal web server uptime is 15 seconds. Thus, with two web servers, we have a complete cycle of 15 + 15 = 30 seconds.

Web Server Uptimes:

Ngnix is not vulnerable to the attack, so its 15-second uptime is not affected.

Apache starts to slow after 8 seconds when the attack is continuous.

Total Uptime: 15 seconds + 8 seconds = 23 seconds, which is **76%** of 30 sec

Web Server Slow Times:

Apache's slow time is during seconds eight through 12, totaling 4 seconds and approximating **14%** of 30 seconds.

Downtimes:

Apache is completely down for seconds 12 through 15, totaling 3 seconds and approximating **10%** of 30 seconds.

Hence, in summary, with MTD, our WordPress application had

76% uptime with no problem at all;
14% uptime with a lag in response time; and
10% downtime.
With the same system without MTD, the application had
13.3% uptime with no problem at all;
6.7% uptime with a lag in response time; and
80% downtime.

Though this attack is in some ways an idealized test case for a technology such as this, it is not trivial, as it represents actual behavior a webserver might encounter under a type of zero day attack. However, a wide variety of attacks must be explored to quantifiably evaluate the usefulness of DARE MTD. We discuss additional testing to be conducted in section VI.

C. Enhancement: Moving Target Defense over SSL/TLS

For security reasons, most of today's web traffic is over an encrypted channel, either SSL or TLS. While most enterprises terminate SSL/TLS at their perimeter, there was a desire to research this aspect of the MTD implementation in order to investigate performance impacts.

The SSL/TLS implementation is fairly straightforward. A 2048-bit asymmetric key was used. The certificate is self-signed, since it was only deployed in a testbed environment. The same certificate and key file are used by both the Apache HTTPD and the Nginx web server. Both web servers were reconfigured to enable SSL/TLS capabilities. The web server rotation (MTD) mechanism was also updated to redirect TCP port 443 (standard SSL/TLS web traffic port) to internal Apache HTTPD TCP port 4482 and internal Nginx TCP port 4481.

As expected, the SSL session key is renegotiated with every web server rotation. However, this process is seamless to the end user and the SSL connection, and the user session (login session) is not interrupted. The result confirms the success of MORE MTD in ensuring application availability across rotation windows [4].

A deep-packet analysis was performed to determine the impact of the SSL session key renegotiation that is forced to occur with every web server rotation. The web server rotation (MTD) occurs at random in a bounded interval between 15 and 60 seconds. The test data were collected on an active user session, which forced an SSL key renegotiation with every web server rotation.

The following tests were conducted:

Test 1 – Client session established. Clicked CTRL-F5 to continuously refresh the browser.

Test 2 – Client session established. Continuously navigated between different WordPress posts to force page to reload. (Note: full page does not reload in WordPress, just the page content.)

Test 3 – Client session established. Continuously switched between the WordPress MTD and WordPress Administration sites, which caused a full-page reload each time.

With every web server rotation, and when the client initiates access to a new web page on the hosted application, a new SSL session key is exchanged. For heavily-trafficked web servers, this exchange may impact performance.

- Test Results
  - Test 1 – amount of time spent on SSL key exchange
  - Min: 0.013990016 sec
  - Max: 0.038951084 sec
  - Mean: 0.027150807 sec
  - About 0.04% of the time is spent on SSL session key exchanges
- Test 2 – amount of time spent on SSL key exchange
  - Min: 0.004898061 sec
  - Max: 0.019973054 sec
  - Mean: 0.0167961424 sec
  - About 0.01% of the time is spent on SSL session key exchanges
- Test 3 – amount of time spent on SSL key exchange
  - Min: 0.001204773 sec
  - Max: 0.017264338 sec
  - Mean: 0.0075265284 sec

- About 0.01% of the time is spent on SSL session key exchanges

Several configuration updates can be made to improve SSL performance. First, SSL can be terminated at the perimeter firewall or load balancer, which will nullify this issue. If this is not possible, there are web server updates that can be made:

- Enable SSL session caching/reuse (requires client browser support).
- Trim SSL cipher support to use fast (and secure) ciphers only in order to increase SSL handshake speed.
- Enable keep-alive to reuse TCP connections.
- Enable caching for static content to improve web application performance.

D. Performance Testing

Goals of performance testing (in descending order of importance):

1. Ensure the performance of the system is not adversely affected by switching.
2. Express performance in requests per second and requests per user by using switching.
3. Determine if we can predict the actual minimum and maximum values for rotation.

1) Adverse Effects of IP Table Switching

To measure the performance of the system, we first look at a single user making a given number of requests. We use Python to send a request and measure the elapsed time for a response to be sent. This is done on an isolated network with two VMs: a testbed and a testing machine. The pseudocode of the script is provided in Fig. 4. Fig. 5 illustrates the elapsed time for the 5000 requests. In the chart we observe spikes, which occur during a switch.

```
MAX = 5000
From 1 to MAX:
  Start = CurrentTimeMillisec()
  Request = PerformRequest('192.168.56.101:80')
  Elapsed = CurrentTimeMillisec() - Start
  Return Elapsed, Request.Text()
```

Fig. 4. Pseudocode for single user elapsed time testing.

The average time for a request/response cycle to complete is 3.01743 milliseconds. We see spikes of around 500 milliseconds shortly after a server switch because the IP Tables are reloaded once a switch occurs. The connection is dropped during an IP Table reload, thus prompting the test script to perform a new request. Similar behavior can be observed in Google Chrome. To prevent this extraneous time, we need to use another mechanism to update IP Tables with minimal latency.

2) Performance of Each Server per User

In this test, the goal is to show that with our implementation, we can still process multiple incoming user requests in a reasonable amount of time when compared to using a single server. The hypothesis is that we will get a measure of throughput that is better than Apache running alone but not as good as Nginx running alone. We simulate five users by running the above script concurrently as five instances; we will also use the Apache Benchmarking tool ("ab").

When using the MTD mechanism, we get an average response elapsed time of 0.8 seconds. With Apache only, we get 0.5 seconds, and with Nginx only, we get 0.009 seconds. With MTD, we do not directly experience the performance benefits of Nginx for all users. We do worse than with Apache simply because of the time spent updated IP Tables. If we can minimize this amount of time, then we can obtain a figure between 0.5 and 0.009. From the combined chart (Fig. 6), we see that the response time is well below 0.5 seconds for most users.

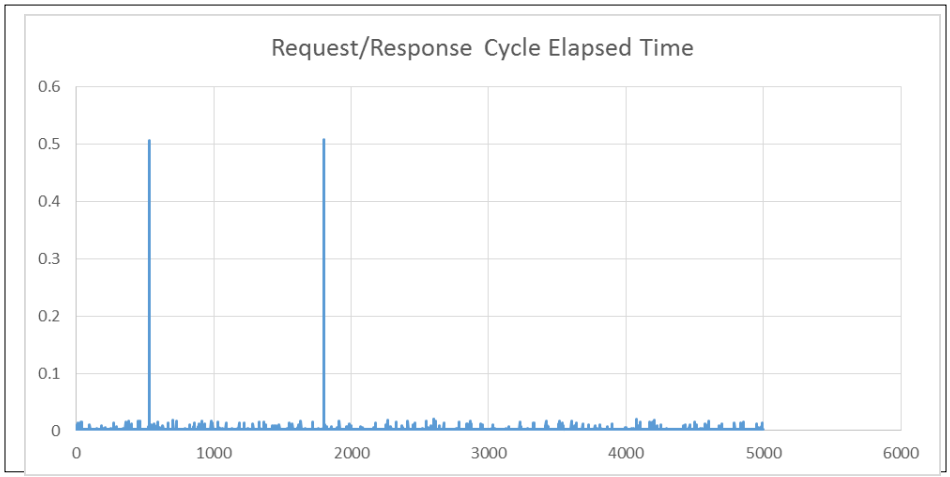

Fig. 5. Elapsed time for 5000 single-user requests; (bottom) elapsed time for 100,000 single-user requests. (Request time in milliseconds is multiplied

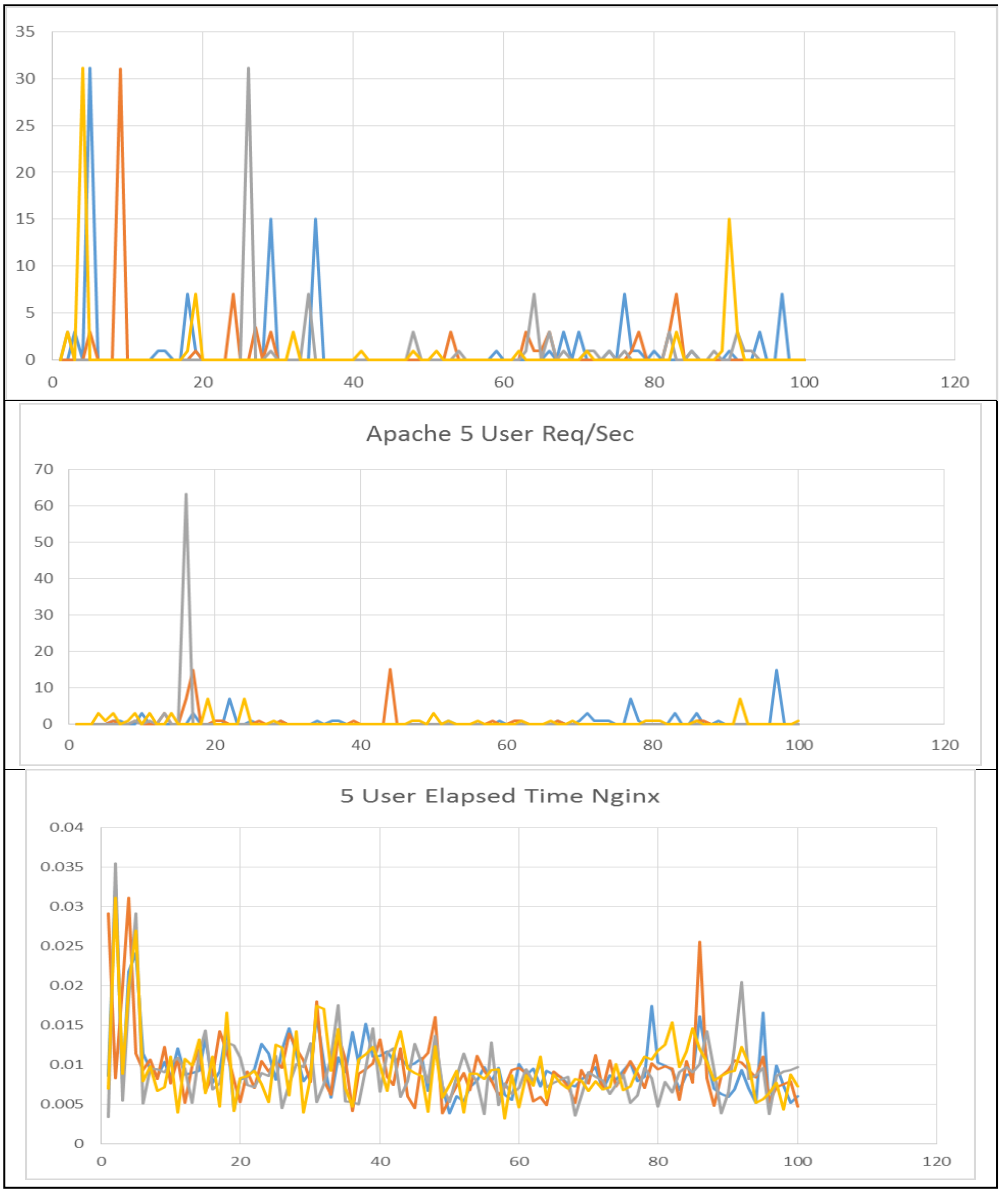

Fig. 6. Five-user elapsed time with (top) MTD, (middle) Apache, and (bottom) Nginx.

3) Interval Prediction

To solve this problem, we first construct a script to measure the time between five server changes. We will use this time to find an average interval. The goal is to see if the average interval corresponds to the configured interval of the MTD implementation. The implementation will select a random sleep time between two values inclusively. Because of the nature of the PRNG function, an attacker will not be able to predict changes in such fine control but will have an idea when a switch is most likely possible.

The solution to prevent the attacker from being able to predict the next change with high probability is to select a large enough interval that we have a large range of randomness but switch enough times to inhibit an ongoing attack. We leave this issue to be addressed as part of future work. Table 2 shows a list of interval switches and the delta time between the changes, which averages 5.4 seconds per switch. The interval in the configuration for MTD during this test was between five and eight seconds inclusive. We can accurately determine a minimum but not a maximum.

TABLE II.

Server Software	Current System Time	Time Delta (ms)	Time Delta (s)
nginx/1.4.0	1450039668573		
Apache/2.2.15 (CentOS)	1450039672181	3608	3.608
nginx/1.4.0	1450039678217	6036	6.036
Apache/2.2.15 (CentOS)	1450039685385	7168	7.168
nginx/1.4.0	1450039690472	5054	5.054
Apache/2.2.15 (CentOS)	1450039695526		
Average Time: 5.4665 sec			

VI. Future Work

Testing conducted demonstrates DARE MTD's ability to improve resilience while increasing attacker uncertainty for a specific type of attacks. This leads to a limited set of promising conclusions, however additional work is needed to quantify DARE MTD's overall effectiveness. Future work should incorporate a wide variety of vulnerability types, such as previously reported vulnerabilities to Apache and Nginx in the Common Vulnerabilities and Exposures (CVE) database maintained by MITRE. These tests should attempt to simulate a variety of attacker skill from "script kiddie" to experienced attacker with full knowledge of the rotation environment details. In particular, the time delta between the execution of a successful exploit and the end of the rotation exposure window should be analyzed for the purpose of minimizing potential consequences of any successful attack.

In addition to attack testing, load testing and subjective user testing should be conducted in order to evaluate potential losses of functionality or productivity due to the introduction of the rotation environment. Attempts to quantify the increased cost of system administration should also be factored into any overall measure of effectiveness.

VII. Conclusions

DARE MTD offers significant promise as a proactive defense against web server application-level vulnerabilities. It succeeds in both the goals of increasing uncertainty (as shown in the VM fingerprinting tests) and increasing resilience (as shown in the exploit mitigation tests). There are a number of unsolved problems with DARE MTD and MTD solutions in general. Though in the present set of experiments, we showed that performance is actually increased over a static Apache web server, performance during rotation remains an area that requires further exploration. Additionally, there are several unanswered questions regarding session maintainability, stateless transport mechanisms such as the user datagram protocol, and overall maintainability with regard to patching and system overhead. However, the conclusion remains that MTD technologies, such as DARE MTD, have the ability to offer significant benefits for protecting high-value targets.

VIII. Acknowledgment

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne. Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

This work is based on research supported by the U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357.

IX. References

[1] U.S. House of Representatives, Subcommittee on Cybersecurity, Infrastructure Protection, and Security Technologies, "Examining the cyber threat to critical infrastructure and the American economy," Washington, DC: U.S. Government Printing Office, 2012.

[2] D. Evans, A. Nguyen-Tuong, and J. Knight, "Effectiveness of moving target defenses," in Moving Target Defense, S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds. New York: Springer, 2011, pp. 29–48.

[3] R. Colbaugh and K. Glass, "Proactive defense for evolving cyber threats," Proc. 2011 IEEE International Conference on Intelligence and Security Informatics, Beijing, July 10–12, 2011, pp. 125–130.

[4] M. Thompson, N. Evans, and V. Kisekka, "Multiple OS rotational environment an implemented moving target defense," Proc. 7th International Symposium on Resilient Control Systems, Denver, CO, Aug. 19–21, 2014, pp. 1–6.

[5] CVE Details, "Secure your Apache server." [Online.] Available: https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-93077/Apache-Http-Server-2.2.15.html

[6] J. B. Hong and D. S. Kim, "Scalable security models for assessing effectiveness of moving target defenses," in 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, GA, June 23–26, 2014, pp. 515–526.

[7] First ACM Workshop on Moving Target Defense (MTD 2014) in conjunction with the 21st ACM Conference on Computer and Communications Security (CCS), Scottsdale, AZ, November 3–7, 2014.

[8] R. Belics, "Out-of-the-box CentOS in VirtualBox (NAT): Network is unreachable," StackExchange: Unix & Linux, 13 June 2013. [Online.] Available: http://unix.stackexchange.com/a/79273. [Accessed 15 October 2015]

[9] P. Schaffner, "CentOS as a Guest OS in VirtualBox," CentOS. [Online.] Available: https://wiki.centos.org/HowTos/Virtualization/VirtualBox/CentOSguest. [Accessed 10 October 2015]

[10] StackExchange, "SELinux: allow httpd to connect to a specific port," serverfault.com, 28 December 2013. [Online.] Available: http://serverfault.com/a/563893. [Accessed 10 October 2015]

[11] NITRD, National Cyber Leap Year Summit 2009 co-chairs' report, networking and information technology research and development. Technical report, National Office for the Federal Networking and Information Technology Research and Development Program, September 16, 2009.

[12] J. F. Meyer, "Defining and evaluating resilience: a performability perspective," presented at PMCCS-9, International Workshop on Performability Modeling of Computer and Communication Systems, Eger, Hungary, Sept. 17, 2009.

[13] C. G. Rieger, "Notional examples and benchmark aspects of a resilient control system," Proc. 3rd International Symposium on Resilient Control Systems, Idaho Falls, ID, Aug. 10–12, 2010, pp. 64–71. a

[14] M. Atighetchi, N. Soule, et al., "The Concept of Attach Surface Reasoning," Intelli 2014, At Sevilla, Spain https://www.researchgate.net/publication/264707910_The_Concept_of_Attack_Surface_Reasoning, [Accessed 23 April].