A rotational moving target defense solution for web servers

by

Steffanie Bisinger

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance (Computer Engineering)

Iowa State University

Ames, Iowa

2018

DEDICATION

This thesis is dedicated to my friends, family, and colleagues who have stood by me and kept me (relatively) sane as I pushed my limits. I am constantly thankful that I did not have to go it alone on this journey.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. Firstly, I would like to thank Cimone Wright-Hamor for working so hard on this project and for all of her help with understanding thesis writing, and Cimone Wright-Hamor, Jeff Neal, and Ben Blakely for collaborating with me on the paper that is to accompany this work. Working with them has taught me so much about both computer security and writing. Without them I would never have found this project.

Second, I would like to thank all those who helped in the editing process, particularly those who were pulled in on short notice.

ABSTRACT

Administrators of web servers fight a constant battle to keep their software current and secured from automated tools that roam the Internet looking for easy targets. With the increasing frequency of patches and the increasing complexity of web servers and the web applications that run on them, this is no easy task. Moving target defense solutions are a class of security solutions that use rotation of services or other strategies to create uncertainty for the attacker. The moving target defense solution presented in this paper rotates between two web servers on a single virtual machine with the goal of increasing the complexity of reliably targeting the software of an individual web server implementation. The evaluation of moving target defense systems, the requirements for such a system to be usable in industry, and the testing platform for our moving target defense solution are explored.

CHAPTER 1. OVERVIEW

1.1 Introduction

As reported by Netcraft in February 2018, there are 1.8 billion publicly-accessible websites on the Internet[4]. According to a report on bot traffic from January 2017, these websites can expect one third of their traffic to come from malicious bots[12] that are programmed to search large address spaces for easy targets (i.e. systems that have not been updated following the release of a vulnerability). To combat new and changing threats, defenses must adapt in parallel with them.

This is the idea behind moving target defense (MTD), a category of defenses that seek to make the defending system harder to hit by rotating or changing some part of the systems configuration. Examples of moving target defenses include changing the layout of the address space of a program or rotating the IP address of a server while keeping legitimate users connected.

The Dynamic Application Rotational Environment (DARE) was a proof of concept implementation of a moving target defense solution developed by Argonne National Laboratory for protecting web servers. Its objectives included being easy to set up and maintain. DARE rotated between two web servers with divergent architectures (i.e., web server daemons) on a single host to create additional diversity[9].

A previous analysis of a DARE implementation examined the ability of this solution to reduce the likelihood of a successful attack on a single server[9]. We present in this paper an improved version of this solution that we refer to as DARE Improved (DIM). We explore the viability of this improved version of DARE for use in a production web server environment by exploring its impact on the ability of an attacker to perform reconnaissance on the web server. The goal of this system is to foil an attacker's attempts to learn the exact web server software and version while maintaining availability to the end users.

1.2 Organization of Thesis

In chapter 2, we examine literature related to the security of web servers to lay the groundwork for formally stating the problem that this thesis seeks to solve. We then discuss literature concerning advances in MTD and system agility to present an overarching survey of the types of solutions in existence. This will aid in understanding some of the approaches that others have applied to similar problems and give a basis for the solution presented here. We will further explore existing methodologies by which MTD solutions are tested and evaluated.

In chapter 3, we present statistics on vulnerabilities in web servers in order to emphasize the burden on system administrators of constantly monitoring for patches, some of which require manual installation. We then present our solution, a moving target defense solution for providing diversity to a web service.

In chapter 4, we explain specifics of the operation of the moving target defense solution along with reasoning for our design choices. Then, we discuss the criteria and methodology used to test this solution. We further discuss the differences between testing on a static website versus a dynamic one, and the ways in which our testing is novel for this research area.

In chapter 5, we present the data produced by our testing regimen and provide an analysis of the data.

In chapter 6, we discuss how the test results relate to the larger picture of MTD. We also discuss the possible benefits of our solution and the situations where our solution would be most beneficial.

In chapter 7, we discuss future work that would improve upon our solution and our testing framework.

CHAPTER 2. REVIEW OF LITERATURE

2.1 Web Server Security

Web servers have become extremely prevalent with upwards of one billion websites as of February of 2018, and, as they have come to host some highly personal and confidential data, are a large focus in the cybersecurity field.[4]

One common method of protecting web applications in particular is to train a web application firewall to protect against certain kinds of attacks and detect certain others. The web application firewall behaved like a wrapper around the web application. These protection mechanisms focused on a combination of user behavior, the source code of the web application, and system behavior to determine what is malicious. As discussed in Prokhorenko et al, such a defense mechanism requires manual setup and configuration both on the part of the organization publishing the rulesets and the part of the system administrator[6]. Thus, a web application firewall takes effort similar to that required to fine tune an intrusion detection system, but potentially provided protection against attacks like cross-site scripting and SQL injection. This was a useful technology for protecting large web applications that take user input. However, the setup time potentially made it more effort than it was worth for smaller applications, particularly those that serve static content.

2.2 Moving Target Defense

Moving target defense is a type of security mechanism that makes changes to aspects of the system in order to create diversity within that system to deter, delay, detect, or prevent attacks. Diversity is artificially introduced with the intention of preventing or mitigating specific types of attacks against the system.

There have been several moving target defense solutions that focused on web servers. An example of this would be NOMAD, which changes the identifiers of elements within HTML to make it difficult for web bots to find the elements of the page they want to interact with, such as the login and password fields of a web form. This solution avoids impacting normal users because it only changes the way that the bot sees the website, changing it syntactically without changing the content or behavior[10].

Another example of a moving target defense solution primarily focused on protecting web servers was the Multiple Operating system Rotational Environment (MORE) MTD developed at Argonne National Laboratory. This used a pool of virtual machines each running a different operating system. MORE MTD rotated which of these virtual machines was actively serving requests. All of the virtual machines used a shared back end database server and a controller that managed the rotation. As nodes are rotated out of the primary role, MORE MTD performs a file integrity check to ensure the system has not been inappropriately modified[8].

The original Dynamic Application Rotational Environment (DARE) paper proposed a solution that also protected web servers using a rotational moving target defense approach. DARE rotated between two web servers (Apache and Nginx) in order to prevent attackers from attacking one server specifically. The tests done in this paper used a Wordpress site and attempted to show the effects of the rotation on availability[9].

Saidane et al proposed a moving target defense strategy for making a "commercial off-the-shelf" web server intrusion tolerant. This idea of intrusion tolerance drew heavily from ideas within the fault tolerant systems area. For example, in assuming that, intrusions would eventually occur much like faults in electrical components that can occur during a system's life cycle. The strategy sought to stop intrusions from creating changes to the system by utilizing redundant servers and a system for controlling the level of redundancy needed[7].

Kewley et al introduced an MTD solution called DYNAT that randomized fields in TCP and UDP packets. This solution protected against reconnaissance for traffic between two computers on the same network[2]. Okhravi notes that this solution can protect against the reconnaissance and access phases of the cyber kill chain and can be used to mitigate scanning[5].

2.3 Categorization and Evaluation of Moving Target Defense

Whereas there are a number of papers [5][7][8][9][10] on various techniques and implementations of moving target defense, relatively fewer papers have been written about how best to evaluate these systems or even what criteria of these systems should be evaluated.

McDaniel et al defined the term *system agility* as the ability of a system to utilize techniques to respond to threats. The paper also presents a system for categorizing agility mechanisms as either wrappers, capabilities, or distributors. The most relevant of these categories to our work is the wrapper category, which the paper defines to be a ”layer that wraps entities such as software or services, to enable agility maneuvers”[3].

We consider DARE, which was discussed in section 2.2 to be an example of the wrapper category because this solution wraps the functionality of Apache and Nginx to prevent attackers from specifically targeting one of the two servers.

Solutions like DARE and MORE could fall under the category of distributors, which McDaniel et al define as ”allowing a single service to be performed on, or distributed to, multiple systems” because both solutions have some load balancing aspects[3]. MORE, in particular, fits this category because it will take potentially compromised virtual machines out of the pool of active virtual machines, allowing the service to continue running in spite of the compromise. The wrapper and distributor categories have the potential to deceive attackers about the nature of the target system.[3] The terminology used in McDaniel et al is useful for differentiating types of moving target defense solutions.

Xu et al explored previous evaluations of moving target defense systems and classified these experimental techniques into two categories:

1. high-level theoretical evaluations based on probability or simulations, and

2. low-level attack-based evaluations.

In this context, high-level refers to statistical models of the entire system; low-level refers to an analysis of data collected from the process in question (ex. CPU or memory usage).

They then posit that there is a gap between these two sets of techniques that can be filled by looking at moving target defense systems on multiple levels of abstraction. They say that many moving target defense papers focused on a single process within the system without considering the impacts to the overall system. They state that other papers focused on higher level concepts without considering the impacts at a lower level. Xu et al also presents their own categories for moving target defense systems: software-based, runtime-based, communication diversification, and dynamic platform[11]. We consider MORE and DARE to be examples of the dynamic platform category because they both exhibit temporal and architectural diversity.

Okhravi et al surveyed a wide array of MTD solutions. Their system categorizes solutions based the type of attack protected against, the step in the cyber kill chain effected, the type of entity being protected, and possible weaknesses in the MTD solution. Okhravi defines their cyber kill chain to have the following steps: reconnaissance, access, exploit development, attack launch, and persistence[5]. They further break down their categories by how much the system must be modified and how much expertise is needed to make those changes.

CHAPTER 3. STATEMENT OF PROBLEM

3.1 Explanation of Problem

Publicly accessible web servers are under constant threat of attack with one third of all visitors being a malicious bot[12]. Bots and automated attack tools look for easy targets, systems that have not been patched for vulnerabilities and can therefore be exploited with minimal effort. Those systems are then stripped of useful data and may be utilized in other attacks. Current wisdom says that the system administrator should just keep up with patches, but often the time between the announcement of a vulnerability and the exploit for that vulnerability being automated and used against systems is quite short[1]. In addition, a system administrator may have many systems to manage that also need to be updated, and this is on top of adding new services to keep up with the demands of the company. Thus, updates to a single service may fall by the wayside.

One solution would be to automate the update process. There is software that tries to accomplish this. For example, Ubuntu, a popular Linux distribution, has an 'unattended-upgrades' package that can be installed to update most of the software on a system without the need of a system administrator. While this may work for certain software, web applications like Wordpress may rely on multiple packages with inter-dependencies, and often these updates break something, forcing the system administrator to do the update manually. According to a report by Kenna Security from 2015, the average gap between the release of a patch and the installation of that patch is over 100 days[1]. An additional solution is needed to protect web servers during that period of time.

3.2 Objectives for a New Solution

Per the above, our objective is to create a more practical solution for protecting web servers from automated attacks against the server software itself. Thus, our goals for this solution are as follows.

3.2.1 Ease of Deployment

Our solution is aimed at helping system administrators who are short on time. Thus, this solution should not take up more of the system administrators time than setting up an auto-update system or setting up a web application firewall. This means that the system will need an installation script, preferably one that only can be run quickly and works across multiple operating systems.

For example, a WAF would potentially be more work than what we consider to be easily deployable. It potentially requires the system administrator to spend extra time configuring the solution. Additionally, if the web application is not configured correctly to operate behind the WAF, the system administrator will get either too much or too little information, or the application may not function as intended.

3.2.2 Minimal Impact to Performance and Availability

A system administrator will not choose to use a solution that negatively impacts their users, as this prevents the system administrator from meeting operational requirements. Therefore, our solution must be lightweight. It is imperative that any rotational aspects of the solution be tested for performance and sustained availability to the user. We define minimal performance impact to mean that the resource usage of the solution is far less than that of the server when at peak CPU and memory load. However, any system administrator interested in using DIM will need to evaluate whether this is acceptable for their environment.

3.2.3 Mitigation of Automated Attacks

The third objective is to mitigate automated attacks by raising the cost to the attacker of performing reconnaissance using port scanning tools against a web server. As a baseline assumption, we do not expect to be able to protect against tools that are specifically meant for targeting rotational solutions (if such tools exist) or against determined attackers who are willing to spend time dissecting the behavior of our solution. Our solution must significantly impair tools like Nmap that expect the services present on a target to remain present during the scanning period.

In particular, we assume that most bots that target web servers will be scanning for a specific version of one web server that the virtual machine is running because they use an exploit specifically for that version of that web server. Thus, preventing a bot from detecting version attributes of its target is the first line of defense against such bots. Failing that, the solution must have the ability to lessen the likelihood that the targeted exploit can be used effectively.

CHAPTER 4. METHODOLOGY

4.1 Implementation

In Chapter 3, we discussed the difficulties system administrators face with regards to keeping servers up to date. We also discussed the goals for a solution to this problem. To solve this problem, we created an application (written in Python) modeled after the DARE solution discussed in our literature review in Chapter 2. We call this new Python implementation DARE Improved (DIM).

DIM improves upon DARE in two main ways. Firstly, DIM utilizes the masquerading functionality of iptables, a popular open-source firewall for the Linux operating system. This functionality passes traffic from TCP port 80 on the external interface of the virtual machine to the servers listening on the internal loopback interface of the machine as though they were on the external interface. This means that our web servers and the applications they are serving are only accessible if they are currently the primary server in the DIM rotation (i.e. the system accepting incoming requests). This forces an attacker to coordinate their attacks to match an exploit with the current primary system parameters. While this may not stop more determined attackers, we will show in Chapters 5 and 6 that this preventative MTD solution decreases the effectiveness of bots that scan for a specific version of a web server across a broad network address space and run a matching exploit against it.

The second area where DIM improves upon DARE is its ease of installation and use. Because DIM uses Pythons argparse library, users can adjust several parameters of DIMs execution including the minimum and maximum rotation times, the ports for the web servers on localhost, the port to serve to on the external interface, and the location to which iptables configurations are saved. DIM also includes a setup script that installs and configures Apache and Nginx (our default web servers) on the machine, taking into account the specific Linux distribution being used. Currently, we support Debian, CentOS, and Ubuntu; it is likely that the script will also work on Fedora since

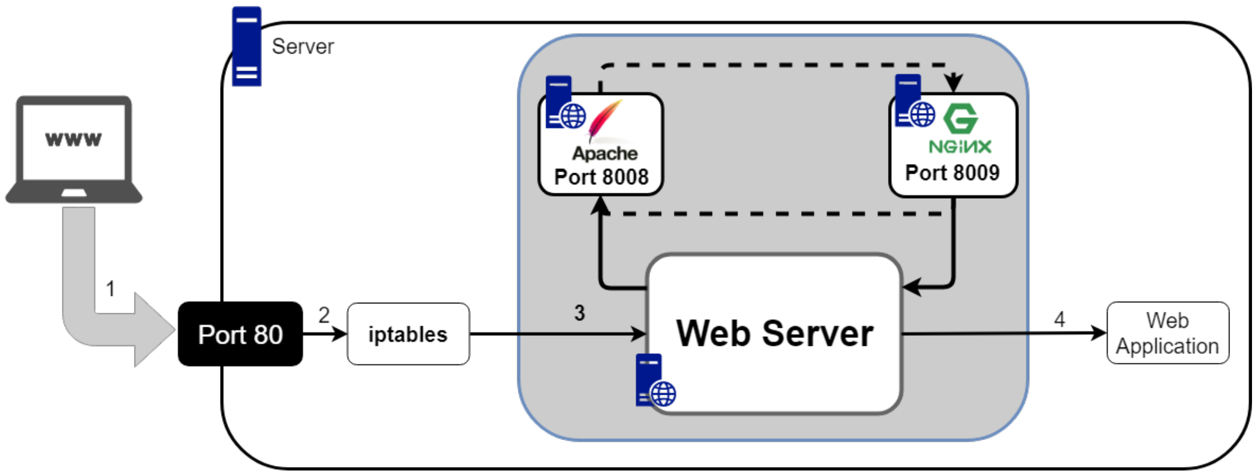

Figure 4.1 An overview of DIM

it is closely related to CentOS, though this has not been tested. It may be possible to adapt our solution to a Windows server provided that server can run iptables, but this is future work.

4.2 Testing

We also improved the processes around DIM by developing a testing framework specifically for evaluating DIM. Our testing framework was created to test three properties of the system:

1. Availability: how often a user will be unable to reach our website due to an in-progress rotation,
2. Performance: how much processing power and memory our solution utilizes, which can disrupt other tasks on the system or slow down responses to requests,
3. the ability to use port scanning tools to determine which server is active to facilitate exploitation.

We use two virtual machines running Apache as the web server and one running Nginx as the web server, as our control instances for each test. We run each test on DIM, with the following test regimens:

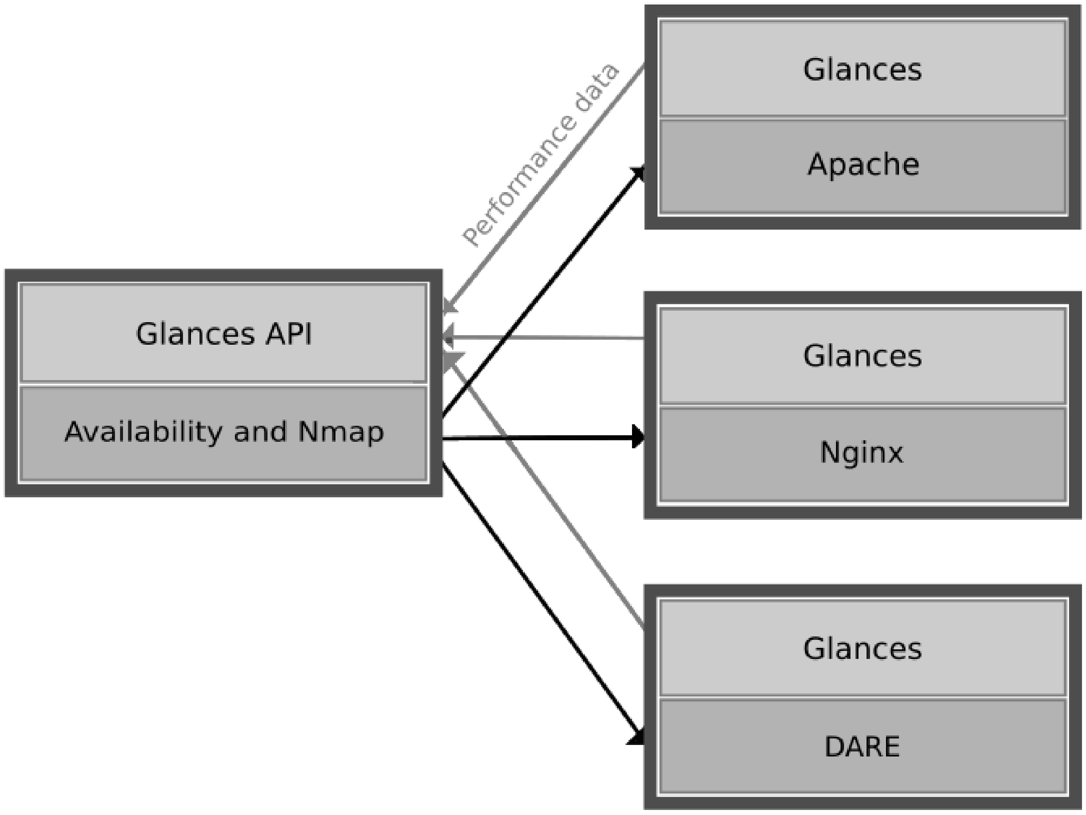

Figure 4.2 The test framework

1. a random rotation time with a minimum rotation time of 1 second and a maximum rotation time of 5 seconds,

2. a static rotation time of 1 second, and

3. a static rotation time of 5 seconds.

4.2.1 Performance

The performance aspect of our testing evaluates whether DIM is causing significant overhead that might impact the ability of the web servers to process requests. This is especially important for complex applications which need a large percentage of the available resources in order to operate effectively.

DIM Research
https://localhost/StaticWebsite.html
This is a static test page for DIM-MTD

Figure 4.3 Diagram of the static testing page

In order to test this, we utilize an open source Python application called Glances that monitors CPU usage, memory, and other statistics in much the same way as the Linux command line tool top. We then use a Python script to export the data collected by Glances as a comma separated value (CSV) file. The CSV can then be processed using the Python library Numpy to obtain a statistical analysis.

As we would expect the system to behave differently when there is little to no load as opposed to the load produced by our other tests, we will also need to run this test as part of our other tests. Thus, we will have a control run with no incoming requests and then we run this test alongside the other categories of tests to give context to the performance of the system.

4.2.2 Availability

We define the availability of the system in terms of how often a user is unable to access the web site hosted. We choose to use this definition because it is the standard in the area of fault tolerant systems, the idea being that small decreases in the failure rate of a system are much more statistically significant than miniscule increases in the success rate. By measuring availability, we

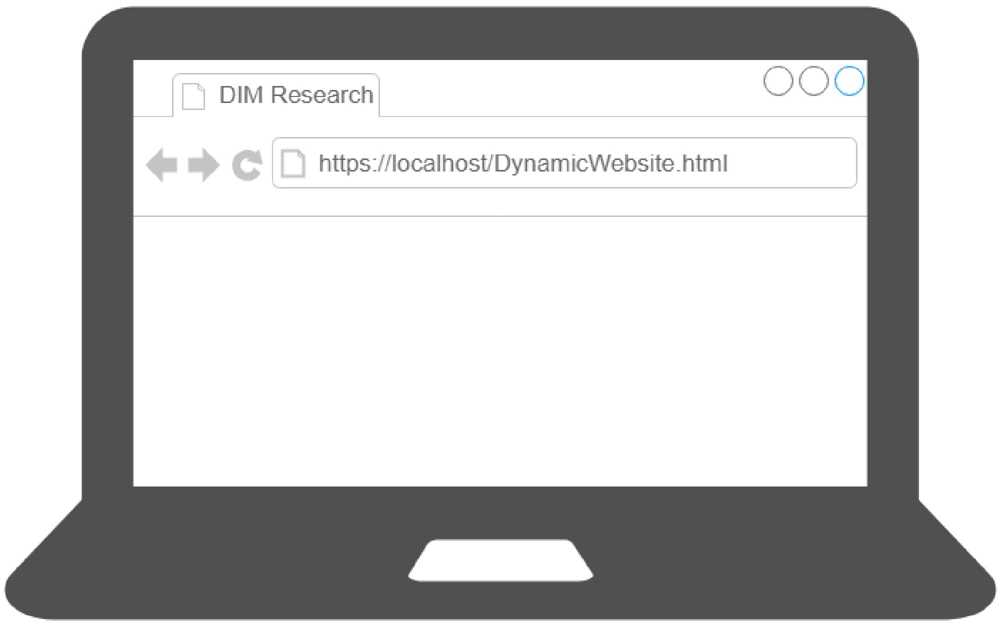

Figure 4.4 Diagram of the dynamic testing page

are able to determine the effect of our solution on the users ability to access the content on our website. This is crucial; if a user cannot use a website it may negatively impact the revenue, reputation, or other assets of the organization running the website.

While other works in this area chose to test availability manually [9] (e.g. by repeatedly pressing the refresh button in a web browser), we have opted for a more automated approach in order to lower the interval between requests and place a more consistent load on the system being tested. This approach allows us to test the system at a lower level (specifically, the Transport and Session layers of the OSI model as opposed to the Application layer) and, because our tests do not rely on a browser, there will be no confusion as to whether or not we are getting a cached version of the page as we might in manual testing.

Our Python script for this test (which, as stated in 4.2.1, also runs our performance testing scripts) sends requests to the server for the index page of the server and looks at a field in the response to determine whether Apache or Nginx sent the response. If neither server is able to respond, as may happen during rotation, this field in our resulting data will be none for this timestamp. In order to correlate rotation times with the data from this script, we also have a script that checks the iptables Network Address Translation (NAT) table to determine which server should be servicing requests at that particular timestamp. By comparing the timestamps in the two data files, we can determine whether dropped requests happened within a rotation time interval and are thus related to the rotation. We can also determine which failed requests occurred outside of a rotation time interval and attribute them back to the appropriate web server.

4.2.3 Fingerprinting

Reconnaissance is typically the first step of any attempt at exploiting a system. One of the primary goals of DIM is to make the system more difficult to identify. Thus, this test is vital to understanding the effects of DIM on the reconnaissance phase of an attack. This step of the attack process can take the form of passive or active scans. Nmap, an open-source port scanning tool, is a popular tool used for active scanning of a system and has many different options for different types of scans. There is also a Python library for interfacing with this powerful tool, which made it ideal for inclusion in our testing framework.

Similar to our other tests, this test is initiated by running a script that repeatedly performs the desired action. In this case, the script repeatedly runs Nmap scans against our test server using a Python library while we monitor the performance of the system. The Nmap script returns a set of comma separated values for each Nmap scan telling us which services it determined to be running on our test server along with timestamps for these values. By matching up these timestamps with our availability testing and the logs produced by DIM, we are able to determine the accuracy of Nmaps returned values.

CHAPTER 5. RESULTS

5.1 Availability

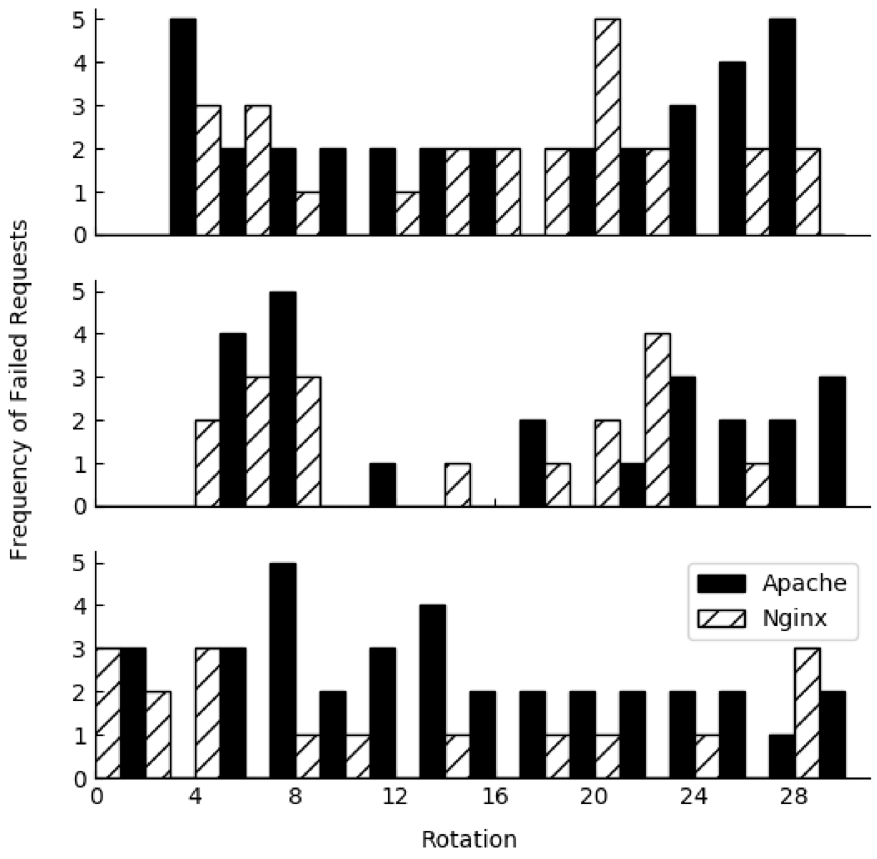

Figure 5.1 Average dropped requests by rotation time

After running the availability tests and averaging the results over several runs, the results of the availability testing show the control systems performed as expected: Nginx and Apache, dropped no packets. By comparison, DIM dropped about 2 percent with the rotation time set to 1 second,

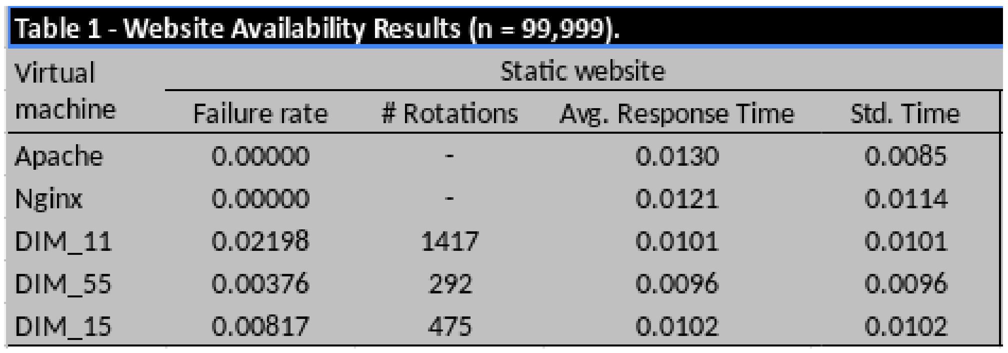

Table 1 - Website Availability Results (n = 99,999).

Virtual machine	Static website			
	Failure rate	# Rotations	Avg. Response Time	Std. Time
Apache	0.00000	-	0.0130	0.0085
Nginx	0.00000	-	0.0121	0.0114
DIM_11	0.02198	1417	0.0101	0.0101
DIM_55	0.00376	292	0.0096	0.0096
DIM_15	0.00817	475	0.0102	0.0102

Figure 5.2 Average dropped requests for test virtual machines

as shown in Figure 5.2. The lowest rate of dropped requests was 0.3 percent when the rotation time was set to 5 seconds. The number of dropped requests is a significant cost in terms of availability for using this solution.

We were further able to break down the test data using the timestamps and determine when the requests were dropped relative to the rotation times. Of the dropped requests, all occurred during the rotation. This implies that most of the dropped requests are the result of modifying the iptables configuration and receiving a request simultaneously. This can be mitigated somewhat by using a higher rotation time. However, a different approach would be needed to eliminate the dropped requests.

5.2 Performance

Figure 5.3 shows baseline usage statistics for each virtual machine. We can see that Nginx has a lower usage than the other virtual machines. This shows us that Nginx is more efficient than Apache. We can also see that the DIM virtual machines are bounded on the top by Apache and on the bottom by Nginx. We can reasonably posit that the DIM virtual machines are in between because Apache is only the active server part of the time. Thus, DIM achieves a balance between the CPU usages of the two web server implementations.

We can see from 5.4 that this balancing effect does not extend to thread counts. DIM raises the number of active threads within the system. This is not unexpected given that the DIM virtual machines are running both web servers and the DIM program itself.

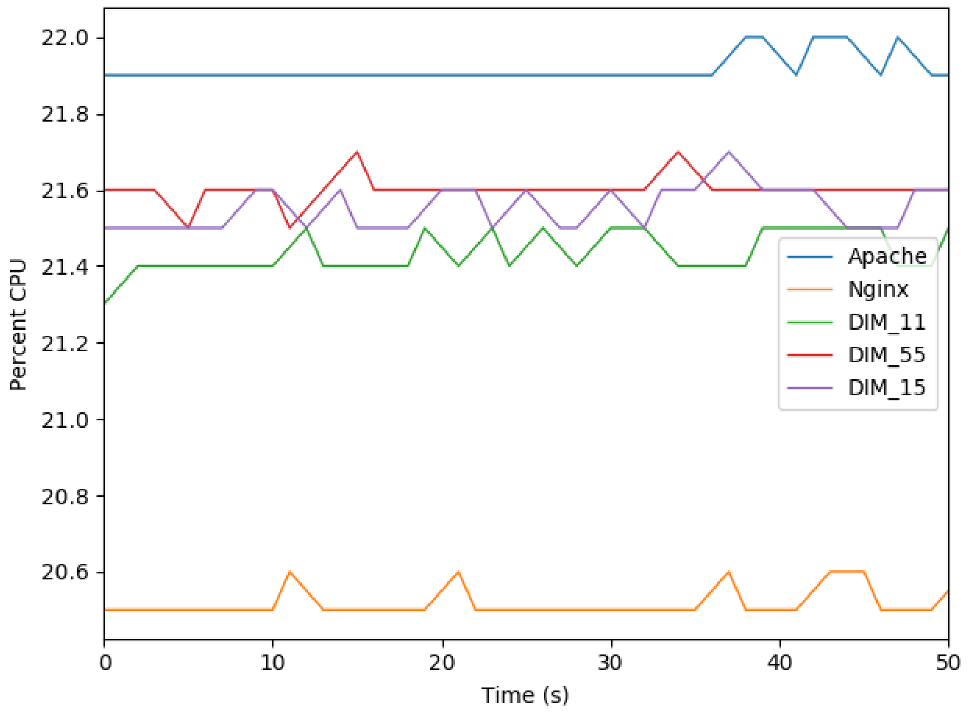

Figure 5.3 Comparison of average CPU usage

5.3 Fingerprinting

After running the fingerprinting test script, which ran the Nmap scan 100 times, the results were processed and averaged. Figure 5.5 shows the average response time for Nmap was the lowest when scanning the Apache server. DIM took the longest to scan on average with an average scan time of 0.7 seconds higher than that of Nginx and 0.3 seconds higher than that of Apache. Nmap was able to identify the control servers with nearly 100 percent accuracy, but identified which server was active on the DIM virtual machines less than 60 percent of the time.

Nmap's ability to identify which server was active on the DIM boxes varied based on the rotation times. The highest accuracy for our test with the static web page was against DIM_55, our virtual machine with the maximum and minimum rotation times both set to five seconds. The highest accuracy was achieved with DIM_11, our virtual machine with a minimum and maximum both set to one second. This may have DIM_15, the virtual machine with a random rotation time between one and five seconds tended to be closer to the times for DIM_11. For the dynamic web page

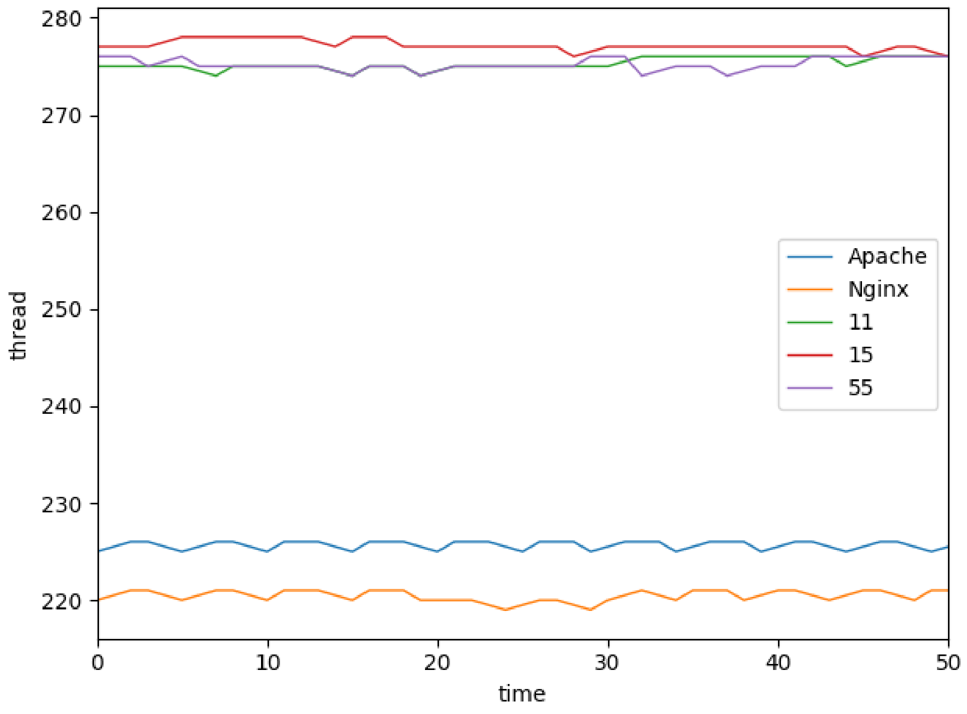

Figure 5.4 Comparison of thread counts

there was much less variation in success rates based on the rotation times. All of the DIM virtual machines had an Nmap success rate under 31 percent with the DIM_15 virtual machine tending towards the success rate of DIM_55. When scanning DIM, Nmap also returned no response 0 to 2 percent of the time. Unfortunately, we cannot compare these results to DARE because that paper[9] did not include specific statistics on how often Nmap returned the correct results.

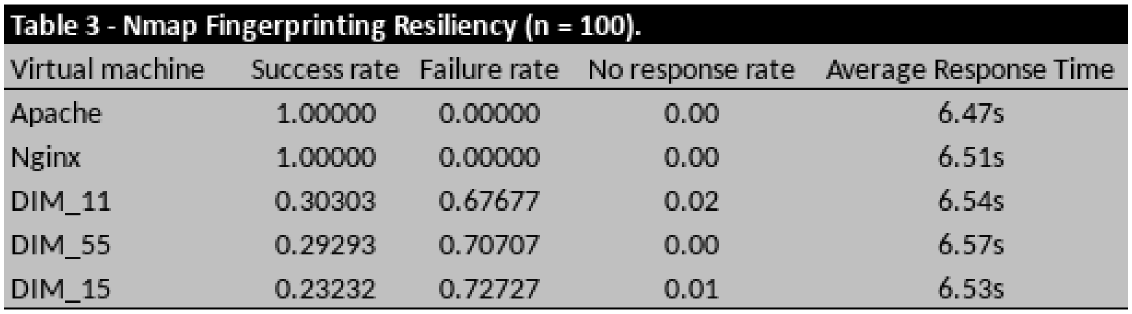

Table 3 - Nmap Fingerprinting Resiliency (n = 100).

Virtual machine	Success rate	Failure rate	No response rate	Average Response Time
Apache	1.00000	0.00000	0.00	6.47s
Nginx	1.00000	0.00000	0.00	6.51s
DIM_11	0.30303	0.67677	0.02	6.54s
DIM_55	0.29293	0.70707	0.00	6.57s
DIM_15	0.23232	0.72727	0.01	6.53s

Figure 5.5 Results of Nmap testing

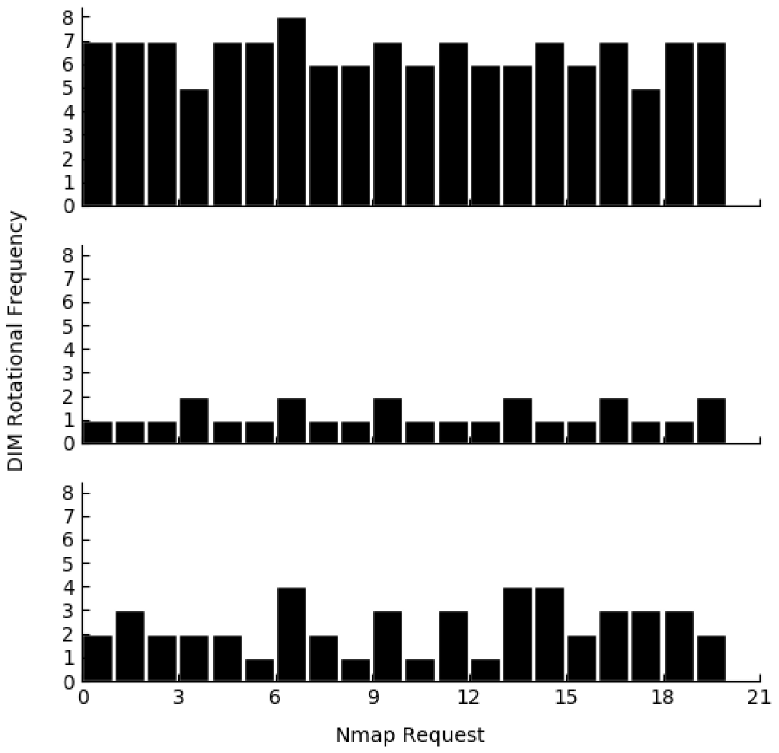

Figure 5.6 Number of DIM rotations per Nmap scan

CHAPTER 6. CONCLUSION

Our goal was to create a solution that would protect web servers. The objectives for this solution, as outlined previously, were that the solution be

1. easy to deploy,
2. have a minimal performance impact, and
3. able to mitigate port scanning by malicious web bots.

The DIM setup script fulfills the first of those objectives by installing and configuring all required packages needed for DIM to run on three distinct Linux distributions using only a single script.

While DIM does use a small amount of CPU (and potentially lessens the CPU usage of Apache), it decreases availability to users significantly. A 0.3 to 2 percent chance that a request is dropped would be particularly detrimental if the hosted web application has real time systems aspect (ex. video streaming, industrial control systems).

However, DIM does cause Nmap to return an incorrect result the majority of the time. A 70 percent failure rate from Nmap means that any attacker will need to take extra time to determine the pattern of rotations using this method. This makes performing reconnaissance against DIM costly. In some situations this may outweigh the cost in availability. However, it will be the responsibility of the system administrator to make that determination for their particular environment.

The testing framework allows evaluation of DIM based on the criteria outlined in Chapter 3. The performance and availability scripts show the impact of DIM on performance while the Nmap script demonstrates the effects of DIM rotations on port scans. The testing framework worked well, and we will likely adapt it to test other MTD solutions.

CHAPTER 7. FUTURE WORKS

While the results of the testing are promising, there is still more work to be done on this solution, and there is still much to explore in the area of moving target defense. In the following sections we posit possible future research directions related to DIM.

7.1 Future Directions for DIM

DIMs current behavior confuses scanning tools, and Nmap in particular, via its rotational capabilities. However, it still currently responds with the active web server, whether that be Nginx or Apache. The next step for this would be to remove that field of the response or to have DIM return a random web server name in that field to further confuse attackers as to what they are looking at.

One of the capabilities that DIM currently lacks that is present in other moving target defense solutions is an intrusion detection mechanism. The author feels that it would be worth considering the application of some form of N-version scheme where the content being hosted by the web server could be easily checked for integrity. Ideally, this addition would also allow DIM to restore the hosted application if one of the servers was exploited, increasing the number of scenarios where DIM would be helpful. The difficulty would be in doing this without adding external libraries that might make DIM more complex to set up.

A further avenue to explore would be to create a similar system that relies on software-defined networking instead of iptables. This would allow the solution to route to the web servers on a per packet basis and might further limit the number of dropped connection and, thus, the effect on availability.

7.2 Future Directions for Testing MTD

One of the simplest next steps for our testing framework would be to increase the load that it is capable of putting on the server being tested. This could be done by either using additional threads or, potentially, by distributing the availability script. One particular technology that could be explored would be Celery, a Python library for creating workers to handle multiple concurrent jobs. While it might increase the complexity of any statistical analysis of the data, this addition would be able to reduce the time between connections to the server being tested and create multiple, simultaneous connections in order to truly simulate user traffic.

A further improvement to the current testing framework might be to add additional scanning tools to the fingerprinting tests to determine whether they react to DIM in the same way as Nmap as not all scanning tools behave the same. This would be necessary in order to say with more certainty that DIM thwarts fingerprinting of web servers.

The ultimate addition to this testing tool kit would an attack component. Unfortunately, we were unable to set up a test to try to exploit a vulnerability in one of the web servers with DIM running as the recent Apache and Nginx vulnerabilities required a good deal of configuration (i.e. compiling from source with certain flags and modifications). In the future, a useful avenue to explore as a proof of concept might be to create a tool that can attack our moving target defense solution as well as other rotational moving target defense solutions.

BIBLIOGRAPHY

[1] Kenna (2015). How the rise in non-targeted attacks has widened the remediation gap.

[2] Kewley, D., Fink, R., Lowry, J., and Dean, M. (2001). Dynamic approaches to thwart adversary intelligence gathering. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 176–185. IEEE.

[3] McDaniel, P., Jaeger, T., La Porta, T. F., Papernot, N., Walls, R. J., Kott, A., Marvel, L., Swami, A., Mohapatra, P., Krishnamurthy, S. V., et al. (2014). Security and science of agility. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 13–19. ACM.

[4] Netcraft (2018). February 2018 web server survey.

[5] Okhravi, H., Rabe, M., Mayberry, T., Leonard, W., Hobson, T., Bigelow, D., and Streilein, W. (2013). Survey of cyber moving target techniques. Technical report, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB.

[6] Prokhorenko, V., Choo, K.-K. R., and Ashman, H. (2016). Web application protection techniques: A taxonomy. *Journal of Network and Computer Applications*, 60:95–112.

[7] Saidane, A., Nicomette, V., and Deswarte, Y. (2009). The design of a generic intrusion-tolerant architecture for web servers. *IEEE Transactions on dependable and secure computing*, 6(1):45–58.

[8] Thompson, M., Evans, N., and Kisekka, V. (2014). Multiple os rotational environment an implemented moving target defense. In *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*, pages 1–6. IEEE.

[9] Thompson, M., Mendolla, M., Muggler, M., and Ike, M. (2016). Dynamic application rotation environment for moving target defense. In *Resilience Week (RWS), 2016*, pages 17–26. IEEE.

[10] Vikram, S., Yang, C., and Gu, G. (2013). Nomad: Towards non-intrusive moving-target defense against web bots. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 55–63. IEEE.

[11] Xu, J., Guo, P., Zhao, M., Erbacher, R. F., Zhu, M., and Liu, P. (2014). Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 97–107. ACM.

[12] Zeifman, I. (2017). Bot traffic report 2016.