Time-Based Controlled Port Hopping: A Moving Target Defense Implementation

Paul Aguilar
Eastern Washington University

Spencer West
Eastern Washington University

Abstract

This project is based around the practice of Moving Target Defense (MTD) in network security. By combining two different MTD approaches, dynamic network and dynamic runtime techniques the hope is to create a layer of obfuscation that makes general reconnaisance more difficult for attackers. Specifically by using a time-based controller to trigger port hopping with a python server. To simulate a separate environment this server was run in a Docker container and then scanned by an attacker in an attempt to gather more information. This design is meant to protect against general reconnaissance attacks, such as port sniffing or active scanning. Ideally, this would be used in conjuction with other defenses to further hinder an attacker whose movements were already restricted by defensive scans or detectors. This was tested on a small scale to show a proof of concept, with a flask server restarting after a random time interval.

1 Background

Moving Target Defense(MTD) is a type of network security designed to implement features that make it harder for attackers to use common techniques by changing the attack surface from a static one to a dynamic one. Two techniques used to introduce this novel design, that are discussed in OpenMTD: A Framework for Efficient Network-Level MTD Evaluation [2] are Port Hopping and Network Address Switching. Port Hopping is the "alternation of ports used by the transport layer protocols, udp and tcp". Several different methods were suggested for this technique, one involving to alternate ports using generated ports from a pool accessible by the admin, rather than using the Well-Known or commonly used registered port. Another method uses a pre-shared key for synchronization, and a function that randomly generates the next port. Finally, another port hopping method revolves around time slots, that determine the next chosen port. While this method doesn't work for UDP it could be adapted for TCP.

On the topic of Port Hopping, if a service was trying to mask itself by having the server change the port that it was listening to, this would help reduce the effectiveness of reconnaissance that a malicious actor will use. Reconnaissance is sometimes the first step of an attack on a service whereby the attacker tries to collect intel on the target.

While both NAS and port hopping follow dynamic network techniques, another class of techniques are dynamic runtime types. This defensive technique works by altering the execution environment. This is what the Mayflies MTD technique uses in Mayflies: A Moving Target Defense Framework for Distributed Systems [1]. The paper explains how their MTD is designed to increase the cost of attack and reduce their success rates. This is done by controlling the system runtime execution using time intervals with a process they call node reincarnations. Explained as the termination of a running node and startup of a new one, possibly on the same or different platform/OS, as it reconnects. This creation process is triggered through proactive monitoring which tries to detect anomalous activity.

The dynamic nature of moving target defenses makes them ideal for combining. As the first implementations mentioned in OpenMTD, utilize both NAS and port hopping, it seems just as plausible to combine the techniques used in both of these papers: time controllers and port hopping.

2 Introduction

With the Internet of Things being present in devices and services from cars to televisions security is sometimes overlooked. Malicious actors will often try to exploit any vulnerability to gain access or information of a system or disrupt services. They might also attempt to exploit a web server through typical ports such as port 80 or port 443. By making the ports that the server listens to as dynamic and random as possible as well as excluding common ports like port 80, then it makes it more difficult for malicious actors to accurately do reconnaissance. A Port-Hopping Technology against Remote Attacks and Its Effectiveness Evaluation explains that "Random port hopping achieves the purpose of confusing attackers by dynamically mapping service ports to ports in the virtual

port space" [3]

It is important that layers of security are implemented to protect devices that connect to the world or networks in order for an organization to provide uninterrupted service. There are many ways for an organization to implement defensive strategies that protect both the product and the consumer. Some of these techniques follow under the umbrella of Moving Target Defense.

Implementing Port Hopping and understanding its practicality on a small network model is the goal of the experiment. By using time-based controls to schedule port jumps the authors of this paper hope to have a server that regularly jumps to the next port and that the previous port is inaccessible while providing the same service on the new port. Ultimately the attacker should have a difficult job at intelligence gathering and having the network and services secure.

3 Threat Model

Our project's threat model is based around a reconnaissance attack. The network structure that the implementation is modeled on is based on the intranet, which is a private network usually used by organizations to share information. Our environment would be a hosted service on a company's local intranet. In the scenario that our time-controlled MTD techniques would be useful is one in which the attacker has somehow gained access within the network and is now beginning that active reconnaissance on the company's internal network. The attacker might be stymied by other defenses on the network, such as dynamic security scans or detectors and can only act in timed intervals. In this way a dynamic time-controlled defense would be useful if the attacker must utilize time-intesnsive scans to avoid detection. In our current implementation, the employees would also be notified by the company of the updated ports, depending on how often they were updated. This way employees would always be able to access the server.

To demonstrate the role of the attacker, one project member used the nmap tool to scan the network in the host virtual machine. As shown in the figure 1, the docker container running the time-based controller and flask server is meant to act as a separate environment that simulates the specific web service on the company intranet. With the server running the group member first scans all open ports using the nmap –p- flag, as they have already gained access to the network the conceit is that they already have the necessary IP information for their scan.

```
nmap -p- 172.17.0.2
```

Once they found an open port, they could attempt to gain more information using that port, for example with a service version detection scan to find the services running on the network:

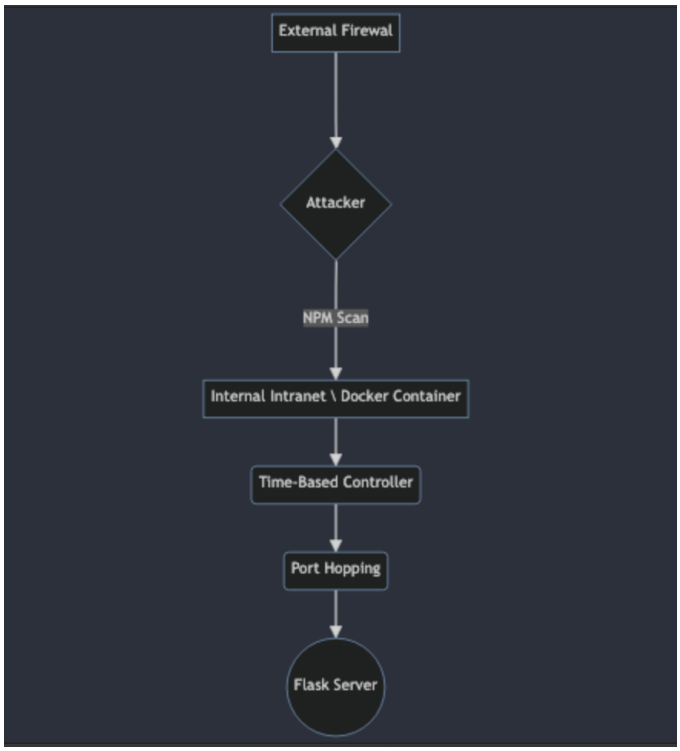

Figure 1: Environment Diagram

```
nmap -sV -p 9097 172.17.0.2
```

There is also an aggressive scan, which would provide more information to the attacker

```
nmap -A -p 9097 172.17.0.2
```

However, this comes at the expense of revealing themselves to the target.

What would most likely be returned after using 'nmap' is that the server is listening on a specific port. The attacker may implement another intelligence gathering technique by using active scanning to analyze the packet responses and see the entire services on the network. To counter the attacker's method of reconnaissance time-based port hoping is implemented so that any information that the attacker has gathered is already obsolete. By having regularly timed port hops that are random then it makes the work of intelligence gathering difficult.

4 System Design

The System design is based around a python flask server whose ports are controlled through a separate python script.

These are used to build a docker container to simulate a separate environment for the attacker to scan. The controller application implements the Moving Target Defense techniques on the flask server. For the purpose of the demonstration an infinite loop is created. Within the loop the script generates a random index for the port list and time interval lists. The port is randomly chosen and passed through to the flask server as it is launched before sleeping for a dynamic period of time. The flask service is then set to wait until the flask server is terminated before the loop begins anew.

Listing 1: Controller application while loop

```
# Python code snippet
while(True):
        port_iterator = random.randint
            (0, 98)
        time_index = random.randint(0,2)

        port=port_list[port_iterator]

        flask=subprocess.Popen(["python3
            ", "fServer.py", str(port)])

        time.sleep(time_delay[time_index
            ])

        flask.terminate()
        flask.wait()
```

As the focus of our project's moving target defense is dynamic runtime/network techniques we chose to implement port hopping on a time-based schedule. These are both designed to be configured by the system admin to increase their dynamism. The time-based schedule is implemented using Python's time library, specifically the sleep method. A list of time intervals in seconds is created, for the purposes of demonstration all three were set to 30 seconds. In a real-world scenario an administrator could populate the list with as many intervals as they wanted of various lengths. A random index is generated using Python's random library's randint method. This index is regenerated every new loop and chooses one of the intervals from the time delay list as the number of seconds the script "sleeps". The sleep period determines how long the flask server is running on any given port before it "hops" to the next one. For this project we decided on one level of random behavior as adding enough complexity for the time delay, however, this time delay list could be implemented in a similar way as the port list, as a list of random intervals generated from a given range. Making the time controller dynamic in both the time intervals generated and the time delay that is chosen for any one loop of the script.

A similar process was used to implement dynamic port hopping in the controller. A list of ports is generated using the randint method, from Python's random library, on a range from 9001-9099. Within the infinite loop of the script a random index is also generated on every loop. This index is used to determine the given port from the port list. That port is then passed in, as a system argument, to a call to run the flask server using the Popen method from Python's subprocess library. The flask server runs as long as the randomly selected sleep interval allows and is then terminated. Using Python's signal library, the flask server is terminated, to close the chosen port. If this is not completed, then all of the used ports remain open. This is not only a security concern, but also prevents the flask server from running on that port if it is randomly selected again. A wait method is used to ensure that the flask server is terminated completely before the loop begins again.

The flask server itself is a simple program that uses the Flask library to run a simple server, on a part based on the port number passed in from the controller script. Currently this displays a message indicating what port it is being run on. However, Flask's render_template could be implemented to run an actual index.html file. In a real-world scenario this would be used to host an internal company website that could be a database, web service, static website, etc., for internal employee use.

To simulate a separate environment that the attacker would be scanning, a Docker image was created to run the python scripts. The image was built using a Dockerfile that imported Ubuntu version 23.1, copied the relevant python files, including the controller script and flask server. It then runs update and upgrade before installing both python3 and python3-flask. Finally, the container runs the controller application which is what starts and stops the flask server. The actual docker container is then created/started using the docker run command with the port flag –p using a range of ports between 9001-9099:9001-9099.

```
$ docker run -p 9001-9099:9001-9099 flask_server:v2
```

This allows for dynamic port hopping without having to continually expose new ports in the Dockerfile. So, rather than continually restarting the container, the controller application regularly restarts the flask server, based on the time delay, within the container.

5 Evaluation

Based on the implementation, Port Hopping was an effective technique that changed the port that the server was listening to. Breaking down the experiment, the implementation was a flask server rendering a webpage that on a time-based interval would change ports. The ports selected were completely random and happened every 30 seconds. The service was then containerized so that it can be run on multiple host machines.

In relation to the threat model the implemented moving target defense techniques did work as expected. However, the bare-bones nature of this proof-of-concept did not provide a

lot for the “attacker” to scan, making it a very quick process. To correctly defend against reconnaissance at this level would mean resetting the ports every few seconds, which would be unwieldy for employees, even more so than the set up already is. The role of the attacker could also be re-designed as a few conceits were allowed to specifically test the port hopping technique. Implementing additional defenses such as network address shuffling would create more work for the attacker and possibly allow for longer time intervals in between resets. So, while this implementation did work in the context of the given threat model, adding additional layers of complexity would make the defense more practical. A more thorough attack scenario with a less straight-forward threat model would also test the usefulness of these defensive techniques a little more realistically.

6 Conclusion

This project was a proof of concept, on a small scale, using a minimal implementation to show the usefulness of combining two dynamic moving target defense techniques. By using a randomly chosen time interval to reset a server to a randomly chosen port, a local intranet web service could be protected from reconnaissance attacks. This project demonstrated that while scanning an environment with tools like NPM, even if a port is found, only so much reconnaissance can be completed before the server was restarted on a new port. Ideally, this concept would be combined with other defensive techniques that waylaid an attacker from continuous scans. The idea being that they would be unable to complete their attack before being discovered by other network security.

If this project was to be iterated on there are a few updates and changes that could be made to make it more reliable. For example, the implementation of cronjobs would be ideal as it would allow for precise timing and resource management, and can be used in Python. Kubernettes should be another consideration to manage cluster nodes when deploying the service as this would allow greater security and administrative controls. Currently the conceit of employees having to be notified of each new port is incredibly cumbersome and would not be incredibly practical in a production environment. To improve this design, a reverse proxy could be added to the model, as shown in figure 2, to listen on one specific port for employees to access as it handles the dynamic port changes of the server. One way to accomplish this would be to configure Nginx as a reverse proxy in its own docker container. This would not only improve the usability of the system, but also its flexibility, making it deployable in more situations. Other features could also be added to the time-based controller’s purview, such as redundancy techniques or even network address shuffling. These additional features would also make the design a stronger defense against more attacks.

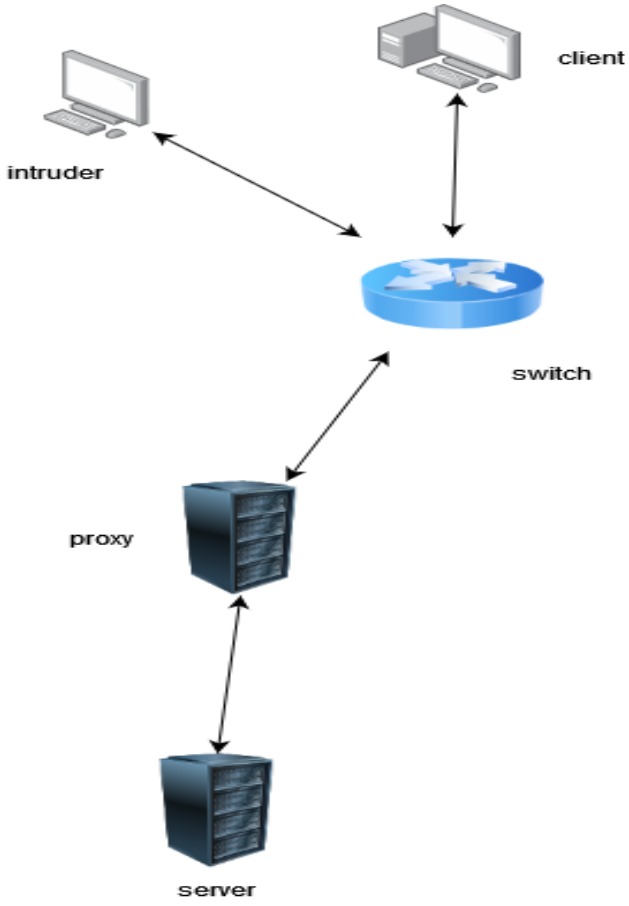

Figure 2: Service with Reverse Proxy added.

References

[1] Noor O Ahmed and Bharat Bhargava. Mayflies: A moving target defense framework for distributed systems. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 59–64, 2016.

[2] Richard Poschinger, Nils Rodday, Raphael Labaca-Castro, and Gabi Dreo Rodosek. Openmtd: A framework for efficient network-level mtd evaluation. In *Proceedings of the 7th ACM Workshop on Moving Target Defense*, pages 31–41, 2020.

[3] Jiajun Yan, Ying Zhou, and Tao Wang. A port-hopping technology against remote attacks and its effectiveness evaluation. *Electronics*, 12(11):2477, 2023.