CY5001Dr. Sierra Fall 2022

Assignment 4.3 MyPyProject

PYTHON PORT SCANNER

MARK FOUDY

Introduction:

For my final project, I decided to enhance the capabilities of the port scanner I coded in assignment 4.2. My goal was to create a command line tool which I could use to reliably scan and enumerate open ports on target computers. I first refactored the UDP scanning function that I created in Assignment 4.2. Once I debugged that process, I moved on to create and expand the script's functionality. Below is a list of the command line options which are built into the script. The PyPortScanner is a port enumeration tool which can be deployed on the command line and used to evaluate open ports. It has functionality to scan both TCP and UDP ports.

Objectives and Justification:

My goal with this project was to create a functioning and operational port scanner that I could feel comfortable using to evaluate open ports on hosts on any network. The limited period I had for completing this project impacted some of my design choices. First, this project was a continuation of the work done in Assignment 4.2. The purpose of this project was to enhance the functionality of the script. As a result, I resolved early in the project to avoid the temptation to write the script using an Object Orientated programming approach. Python is a multi-paradigm programming language, meaning that it can be used for both structured programming such as this script and object-orientated programing. It also has an extensive library of modules which allow the coder to use Python for a variety of purposes. In fact, it is common to see Python utilized extensively by cybersecurity professionals, network engineers and data scientists

Roadblocks Found:

The first and most critical roadblock I encountered in writing this program was getting the UDP scanner to work correctly. The difference between UDP and TCP is considerable. TCP is a connection-oriented protocol which provides error correction and proof of reception. In terms of designing a port scanner, all that is needed to log open TCP ports is to receive a response back from the target. In addition to the uncomplicated nature of TCP connect, I was able to benefit from using the Berkeley Socket API which made socket creation easier to implement using Python.

UDP as a protocol differs from TCP, because UDP is not connection-orientated and has no mechanism to guarantee delivery of packet transmission. UDP sends datagrams which do not contain the capacity to check for errors or ensure transmission. When a UDP port is open, a target computer will often respond by passing the UDP connection attempt up to the application layer. It does not have any capacity built in to respond to the sender or make any other acknowledgements of successful transmission. Because UDP is a connection-less transport protocol my UDP scanner needed to account for datagram loss and have a capacity to retransmit to ensure delivery. Unlike the TCP, a UDP port is considered open when it does not respond to a transmission attempt. I spent several days trying to get the UDP scan functions to work properly. There are not many python UDP scanners available to consult, and the ones I found were coded using Python's OOP capacity. I am thankful that Dr. Sierra was willing to share his solution to the problem with the class. This enabled me to focus on adding to the Python script's additional functionality and implement some of the suggestions made in the Final Project Specification. I am incredibly happy with the project now that it is completed. I am

proud of the work I did, and I feel like this was a wonderful opportunity for me to get more experience doing socket programming. In assignment 4.2, I compared my scanner to Nmap. Nmap is an opensource piece of software which is the industry standard to use for host enumeration and port scanning. I had hoped to gain some clarity by viewing Nmap's source code. However, Nmap is coded in C++ and thus has a different design approach which makes comparisons inappropriate.

The aim of this project was for me to modify the Python Port Scanner from assignment 4.2 and give it additional capabilities. On top of that, I endeavored to tighten up the UDP scan feature and try to get it working. I spent hours reworking the UDP scan feature and wrote about five distinct functions, each coming at the problem a little differently. I finally found something that worked after sitting in the lecture and having Dr. Sierra explain how to make the scan work. The UDP port scanner was a definite roadblock. As I mentioned in my assignment 4.2, UDP scans are quite different from TCP scans. TCP scanning is straightforward since all that is required is the initiation of the three-way handshake. However, because UDP is connection-less, it is more difficult to ascertain an open, filtered, or closed port. Thankfully, I was able to get around this complexity through implementing Dr. Sierra's solution to this problem.

Adding additional Functionality through providing Port Information to the user.

I am very happy with the way that I was able to implement the option to print port information to console after each scan. I also built into the scanner the option to specify where that information could be stored in the form of a text file. In order to accomplish this task, I created two JSON files which I used to hold either the UDP or TCP port information. I found it difficult

to use Python's print function to save port information to a file without the output including some strange characters. As a work around, I used the write function which I included after each print statement enabling the port information to be both displayed to the user on the command line and also written to a file if specified by user input.

Development Explanations:

'-H' allows the user to specify either an IP address or a domain name of a target to be scanned.

```
mark@CY5001:~/FinalProject$ ./pyportscan -H 172.17.0.2
```

'-f' allows the user to provide a list of IP addresses or domain names to be scanned in series.

```
mark@CY5001:~/FinalProject$ ./pyportscan -f IPlist.txt
```

'-p' allows user to provide a comma-separated list of ports to be scanned.

```
mark@CY5001:~/FinalProject$ ./pyportscan  -p 20-23,80,135-139,386
```

'-u' allows the script to run only a UDP scan on specific ports selected by user

```
mark@CY5001:~/FinalProject$ ./pyportscan -u 22,53
```

'-t' allows the script to run only a TCP scan on specific ports selected by user.

```
mark@CY5001:~/FinalProject$ ./pyportscan  -t 1000-2000
```

'o' allows the user to specify a filename where output scan data can be stored and viewed after execution.

```
mark@CY5001:~/FinalProject$ ./pyportscan -o output.txt
```

Each of these options can be used together or in combination with others:

```
mark@CY5001:~/FinalProject$ ./pyportscan -f IPList.txt -p 20-23,80,135-139,386
-o output.txt
```

Potential Future Improvements:

There are several improvements I could make to enhance this script. In terms of goals, I think the Nmap port scanner is a good model of what kinds of expansion are necessary. I am not comparing my script to Nmap. Nmap is the most frequent utility I use when diagnosing network problems on my home network. The functionality built into Nmap over the years comes from user input and experiences using it in the wild. I think one of the easiest improvements on this script I could make would be to have more user input on how many times the script is initiated over a given port and how long to wait before re-scan. These were improvements suggested in the final quiz, and they are the most feasible and most useful improvements that could be made. The necessity of these enhancements stems from the fact that system administrators and network administrators do not like people scanning their network for open ports. As a result, a lot of firewalls as well as intrusion detection services are engineered to identify and isolate nodes which are scanning a host's ports. Nmap has an even more robust mechanism to thwart IDS systems through its use of complicated evasion techniques.

As the Nmap book states, "In addition to restricting network activity, companies are increasingly monitoring traffic with intrusion detection systems (IDS). All the major IDSs ship with rules designed to detect Nmap scans because scans are sometimes a precursor to attacks. Many of these products have recently morphed into intrusion *prevention* systems (IPS) that actively block traffic deemed malicious. Unfortunately, for network administrators and IDS vendors, reliably detecting bad intentions by analyzing packet data is a tough problem. Attackers with patience, skill, and the help of certain Nmap options can usually pass by IDSs undetected. Meanwhile, administrators must cope with large numbers of false positive results where innocent activity is misdiagnosed and alerted on or blocked."[1]

Regardless of the direction I go with this scanner, I think I will spend a significant more time focusing on becoming more familiar with Nmap evasion and spoofing techniques. There

[1]Lynn, Gordon ***Nmap Network Scanning***, on page 257. This resource can also be accessed for free at: https://nmap.org/book/man-bypass-firewalls-ids.html

are a considerable number of avenues to pursue specifically with the Nmap Scripting Engine (NSE) which allows for modification and creation of scripts for the Nmap utility. I recently purchased a book on Lua to become more familiar with that language since it is the language used for Nmap scripting.

Conclusion:

The addition made to the Python Port Scanner over the last week has really elevated the script in my opinion. This was only possible with the encouragement and guidance of Dr. Sierra and the TAs. I would like to thank Dr. Jose Sierra and the TAs Ramamurty Kaushik Vaida and Nidhi Patel. I really appreciate the feedback I got from Kaushik regarding my completed work and the material. This is my first class in the Cybersecurity Graduate Program, and I have enjoyed this class more than any other CS class I have taken. I look forward to hopefully working with you all in the near future.