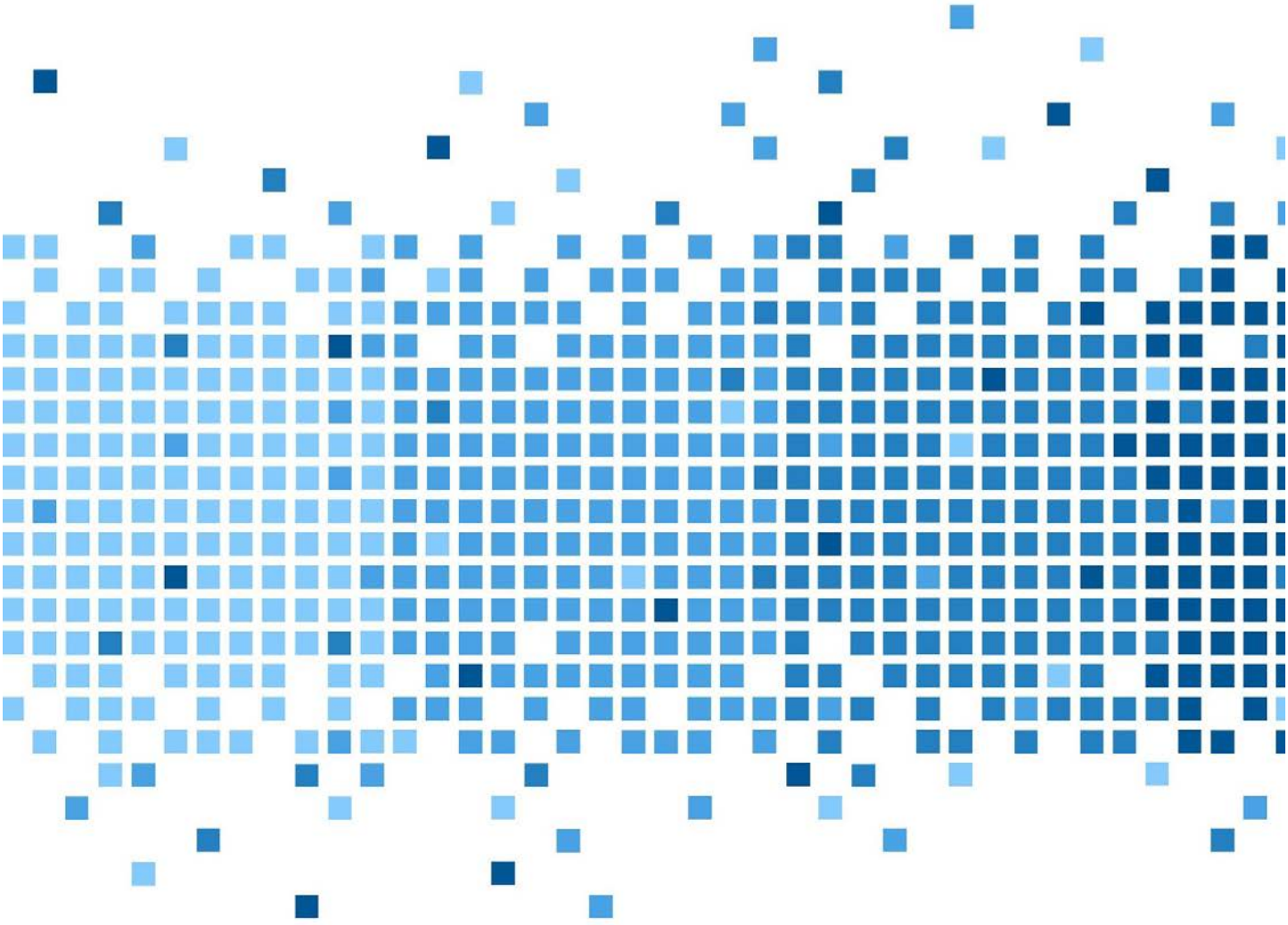
CHAMPLAIN COLLEGE | LCDI Leahy Center for Digital Investigation

Network Scanning



175 Lakeside Ave, Room 300A
Burlington, Vermont 05401
Phone: (802)865-5744
Fax: (802)865-6446
http://www.lcdi.champlain.edu

5/16/2018

Disclaimer:

This document contains information based on research that has been gathered by employee(s) of The Senator Patrick Leahy Center for Digital Investigation (LCDI). The data contained in this project is submitted voluntarily and is unaudited. Every effort has been made by LCDI to assure the accuracy and reliability of the data contained in this report. However, LCDI nor any of our employees make no representation, warranty or guarantee in connection with this report and hereby expressly disclaims any liability or responsibility for loss or damage resulting from use of this data. Information in this report can be downloaded and redistributed by any person or persons. Any redistribution must maintain the LCDI logo and any references from this report must be properly annotated.

Contents

Introduction

Security is vital in the tech world, and nothing is more important than network security. For this reason, our team has been making advancements in the protection of the Leahy Center for Digital Investigation (LCDI) network. Our Network Scanning team has worked diligently to ensure those under the LCDI network are protected from unexpected threats. In today's world, networks can be vulnerable to anything from viruses and worms, to the dreaded DDoS attack. While threats such as these may seem less threatening, they can certainly do a significant amount of damage. Countless hours of research, an abundance of software, multiple Raspberry Pi's, and custom designed scripts helped us achieve our overall goal of assessing and protecting the LCDI network.

Background

A good portion of the information we found came from others who have successfully scanned a network before. From their experiences, we gained knowledge on how to make our own version of a network scanner. As expected, those who have done this already use a variety of different pieces, but the overall idea is the same. Everything from blogs to websites dedicated to this type of technology assisted us in completing our goal. We began by researching all things we would need to complete our task. The information gathered in our group document consisted of basic and high level functions in Python; how to incorporate third-party libraries into our software, the pros and cons of using different single board computers, the commands within Nmap, what ports to scan on the networks, and how to execute scripts on our devices. All of this information contributed to our final goal of scanning and testing the LCDI network.

Purpose and Scope

Aside from our overall purpose, we had smaller goals that had to be achieved first. Obtaining our devices, writing scripts that allowed us to scan, and testing our scanner on our own Pi servers were some of the many goals we focused on. Once it was time, we put all our energy towards scanning the LCDI network. The final scan helped us understand what our network consists of and how we can protect it from future threats.
Research Questions

1. Can we effectively scan a network using a Raspberry Pi?
2. What will we discover by scanning our network?
3. Is there anything we can do to make our scanner more efficient?

Terminology

Bash - a Unix shell and command processor that typically runs in a text window, where the user types commands that cause actions.

Commands - a text interface for your computer. A program takes in commands, which it passes on to the computer's operating system to run.

CPU - is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Crontab - a time-based job scheduler in Unix-like computer operating systems.

DDoS - a cyber-attack where the perpetrator seeks to make a machine or network unavailable by temporarily or indefinitely disrupting services of a host connected to the internet.

Ethernet Cable - a cable that connects devices together within a local area network, like PCs, routers, and switches.

File Server - a device that controls access to separately stored files, as part of a multiuser system.

GPU - a programmable logic chip (processor) specialized for display functions.

HDMI Cable - a cable used for transmitting uncompressed video data and compressed / uncompressed digital audio data from a source device to a monitor, projector, etc.

Kernel - a computer program that is the core of a computer's operating system, with complete control over everything in the system.

LCDI - Leahy Center for Digital Investigation

Library - a collection of precompiled routines that a program can use.

Linux - open-source software operating systems built around the Linux Kernel (see "Kernel" definition)

Malware - software that is intended to damage or disable computers and computer systems

Netaddr - a network address manipulation library for python

Network - a set of computers connected together for the purpose of sharing resources.

Nmap - a security scanner used to discover hosts and services on a computer network, thus building a "map" of the network.

Packets - a formatted unit of data carried by a packet-switched network. A packet consists of control information and user data, which is also known as the payload.

Packet Switching - digital network transmission process in which data is broken into suitably-sized pieces or blocks for fast and efficient transfer via different network devices.

Port - an interface between the computer and other computers or peripheral devices. A port generally refers to the part of connection available for connection between one computer to peripherals like input and output ones.

Python - an interpreted high-level programming language for general purpose programming.

Python-Libnmap - a python library enabling python developers to manipulate Nmap process and data.

RAM - a form of computer data storage that stores data and machine code currently being used.

Raspberry Pi - a low cost, credit-card sized series of small single-board computers.

Script - a computer program written in a scripting language.

SD - an ultra-small flash memory card designed to provide high-capacity memory in a small size.

Server - a computer program or a device that provides functionality for other programs or devices, called "clients".

Single Board Computer - a complete computer built on a single circuit board, with microprocessors, memory, input/output and other features required of a functional computer.

Socket - one endpoint of a two-way communication link between two programs running on the network.

SSH - a cryptographic network protocol for operating network services securely over an unsecured network.

Switch - a device used to connect devices together on a computer network by using packet switching (see packet switching) to receive, process, and forward data to the destination device.

Unix - a widely used multiuser operating system.

Virus - a piece of code that is capable of copying itself and typically has a detrimental effect such as corrupting the system or destroying data.

VMware - a service that provides cloud computing and platform virtualization software and services.

Web Server - a system that delivers content or services to end users over the internet.

Worm - a standalone malware computer program that replicates itself in order to spread to other computers.

Methodology and Methods

Our plan coming into this project was mapped out in the beginning to make it easier for us to stay on track as time passes. The plan consisted of a Raspberry Pi 3 possessing a script designed to scan the ports on one or multiple hosts that we would test against the networks. Before we could scan the LCDI network, we needed to run practice tests to see if our Nmap scanning script worked properly. This is where we implemented three other Raspberry Pi's. Our team decided to use the Pi's to setup three servers; a file server, a web server, and a SSH server. This enables us to see how well our script functions, and from this we can see if there are changes that must be made. By having a test server, we are able to test our script in a controlled environment, which allows us to know exactly what is not working in our script and what we need to change. We used a variety of Nmap scanning commands within our custom script. Being both successful and efficient is what we desired for in the test scans so that we could move on to the main goal, scanning the LCDI network.

Of course with achieving any goal, there will be obstacles that must be overcome. Throughout our project we pushed past issues with writing the script and setting up the servers. We successfully completed these tasks and focused on issues that may arise when scanning the LCDI. Will there be aspects unique to the LCDI network that did not appear on our Pi servers? Will our script be able to handle a more complex network? How efficient will our scanner be on the network? As problems appeared, our team would quickly resolve the issue and get back to completing our goals.

Equipment Used

Hardware:

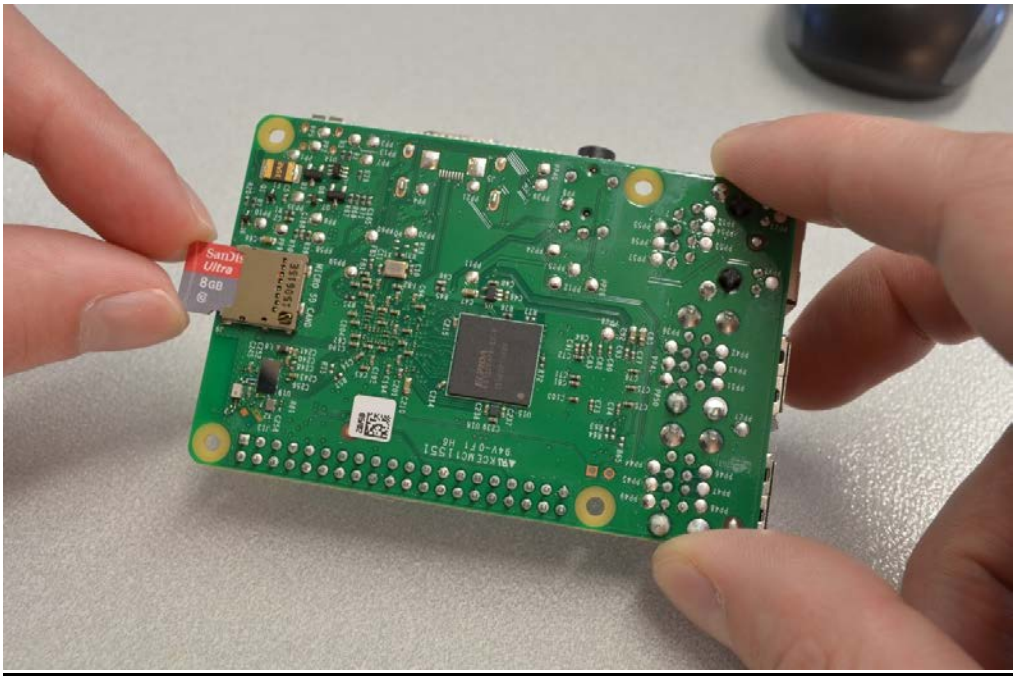
The most important piece of hardware is the Raspberry Pi 3, the device running the script. Despite its small body, the Pi 3 has a Quad Cortex A53 CPU, processing at 1.2GHz. Its GPU is a 400MHz VideoCore IV and can store up to 1GB SDRAM, using a micro-SD. Along with this it contains both video and audio output. This device outperforms its predecessor and is well worth the price of $35.

The other Raspberry Pi's involved in this project were Pi 2s. This older Pi model shares a similar design as its replacement, but is less powerful. This style Pi possesses a Quad Cortex A7 @ 900MHz CPU and a 250MHz VideoCore IV GPU. The Pi 2 has 1GB SDRAM, also using a micro-SD. Despite being less powerful as the scripting Pi, it works well as a server Pi.

Software:

Nmap is the service that was vital to this project. This security scanner allows us to discover hosts and services on a network. The information gathered by this program includes, what hosts are on a network, the services running on the specific port, the type of filters or firewalls that are being utilized, the operating systems, etc.

Another important program that proved useful is Python. This service is an interpreted high-level programming language for general purpose programming. Python is what we used to write the scripts that would be used on the Raspberry Pi 3.



(BC-Robotics.com)

Table 1:Physical Devices

Device	Version	Comments
Raspberry Pi (x 3)	Model 3 | Raspbian	Used to run the script against a network
Raspberry Pi (x 2)	Model 2 | Raspbian	Used as testing networks (file server, web server, SSH server)
Netgear ProSAFE Plus Switch	GS105Ev2	
C&E High Speed HDMI Cable	N / A	Used to connect the Raspberry Pi's to a monitor.
4 Ethernet CAT 5 Cables	N / A	Used to connect the Pis to the switch
TP-Link	TL-WR841N	Used to create a network for the Raspberry Pi's

Table 2: Software

Software	Version	Comments
Nmap	V 7.70	Software that runs scans on a IP's ports
VMware vSphere Client	V 6.0.0	Manages and runs virtual machine(s)
Red Hat Enterprise Linux 7 (64-bit)	V 11	Virtual Machine used to test script functionality before putting it on Pi's
Python	V 3.6	Programming language used to create script and was edited in Python IDLE
Netifaces	V 0.10.16	Python third party library that allows enables us to get the IP address of the Pi

		easily
Netaddr	V 0.7.19	Python third party library that allows us to easily convert IP data such as subnet masks
Python-libnmap	V 0.7.0	Python third party library that allows users to utilize Nmap in Python scripting
Raspbian	2017-12-01	Raspbian is very well documented and the native OS for the Pi, which is why it is being used for this project.

Data Collection

How will you be collecting your data? Need to link the title of the charts and tables in your paragraphs to be to the actual chart/table in your paper.

Data will be collected through the Pi's, which will then be outputted to the user. The overall process starts with an individual plugging the scripting Pi into a host. From here, the Pi will automatically begin to execute the specified commands that the team chose to incorporate in the scan. Different scans will generate unique sets of data, so the information that is extracted will be different based on what scans are performed. When it comes to the actual extract, we decided to have an email sent out to the user. This email will contain the data that was discovered and collected by the scanner. For example, when we run the scan on our server Pis, it should come up with 5 different IP addresses that are up. In this case, the ports open depend on the type of server. An example of this is that on the SSH server, only port 22 should be open. The reason why there are five results and not four is because the scanner also scans the switch. The data collected by the scan performed by the team can be found Appendix 1.

Analysis

At the beginning of the project, our team set the goal of having two main tests for our scanning Pi. The first test that we needed to complete was the scan of our three server Pi's. As stated before, we used the three Pi's to create a file server, a SSH server, and a web server. The way this worked was by utilizing a switch. By using a switch, we could connect each of the servers together, allowing us to scan each of them with ease. Once this test was successful and we fixed all the issues we had, we could move on to the next big step, scanning the LCDI network.

Now that we have successfully completed a variety of scans, we can look back on our research questions and see if we answered them. We believe that we accomplished our original goal. We were able to fully automate the scan, meaning the script was able to get the IP address (including the subnet mask) without any user input. Adding to this, we were able to automate the scan to run every "n" amount of hours/minutes/seconds. In our test scans, this time was changed depending on the scan. When we ran it on the Pi's, we did it every 5 minutes, whereas, when we scanned the network, it ran every day because we knew that

the scans would take longer. Our final scan on the LCDI network was set to run every hour, because our final version ran for around 2 and a half hours. We also automated one NMAP command.

Results

What you did and what you found. It is likely you will have multiple headings here, what they are will depend on your research. Once again it is often easiest to present your data in tabular form.

Pi Tests

The network constructed of Pi's ended up being very successful. In the early stages of scanning this network, we ran into many problems. We had issues with getting the script to function properly. We also had problems, later down the road, with some of our servers not running. For example, we were running a scan on the server over the weekend and, at some point during the weekend, our web server crashed. We had to reimage it during out next shift in order for it to work. Also, our SSH server wasn't working. When we changed the static IP so that all our servers were on the same host, we forgot to uncomment our change in the file it was contained in. When we looked at the results from that weekend, the scans were successful except for the fact that they were missing two servers due to them not functioning properly. After we fixed the servers, however, the script ran beautifully. The nice thing about having a controlled test environment using Pi servers is that we were able to troubleshoot our code easily. We knew what services were running on each host, allowing us to know what ports should and should not be open. This allowed us to tell if the script was finding the right ports. We were also able to tell if there was something wrong with our server Pi's because, if their static IP address was not listed, we knew that there was something wrong with that Pi. There was also an issue with crontab when we first set it up. Crontab does not run the commands from bash, it runs them from sh. Once we found this out we changed it to the correct run path and it was fixed. We also tried sending the results to an email, but was not able to get it to work. Because of this, we chose to put the results in a text file that can be read on the Pi or be recovered off of it. Despite the setbacks, the Pi network allowed us to address any problems that we were having, and through this we were able to conquer all issues. In the end, we completed an abundance of scans and collected the amount of data that we were looking for.

LCDI Tests

We ran our script twice on the network using our Pi scanner. The first time, our scanner contained the commands "-sV" and "-p-". "-sV" tries to find the service that is running on a port, i.e if the scanner detects that port 80 is open then it will try to find the software that is running on that port. "-p-" scans all 65535 possible ports. As previously mentioned, our first scan had very mixed results. We still are not sure why the scan failed the majority of the time and, because of our time frame, we knew that we could not find a fix for the problem. Our solution was that we took "-p-" out of our script. This made it run faster because it was not trying to scan every single port just the most common ones. We found that by making this change, our scanner ran much faster and was a lot more successful. We ran the scanner a second time on the network with this change, and found that 42 out of 45 scans were successful, the failures being empty files and not errors in the script. This scan took place over two days, we started it during our Wednesday shift and checked it Friday. It scanned 65,536 IP addresses on the network and our results display those addresses that are up. We also were able to

make it run every hour without user input, which is why our scan ran that many times over a two day period. Overall, we were successful in scanning the LCDI network.

Conclusion

1. Can we effectively scan a network using a Raspberry Pi?

   Yes, we can. Raspberry Pi's work like small computers. Because of this, they have a capability to do almost anything that a desktop/laptop can do, to an extent. They are small, meaning that they do not contain nearly the amount of equipment that a desktop/laptop has in order to make it fast. However, they are small enough to be portable, which makes them ideal in this situation. After many revisions to our script, we were able to make a scanning Pi that can efficiently scan a network once plugged into it via ethernet cable.

   network. See Scan Times for more information on how long different scans took to complete.

2. What will we discover by scanning our network?

   From the scans we ran, we could see different things about the ports that were associated with the network. Each port is used for a different service. This allows us to see what services are being used on a network by looking at what ports are open. This is extremely helpful when dealing with security because if a port is open that is associated to a service you are not using, it means that the network may not be secure. See Figure 1 for an example of one of our scans.

3. Is there anything we can do to make our scanner more efficient?

   Throughout the entire project, we were trying to find more ways to improve the speed and accuracy of the scan. To do so, we worked on changing the script being run on the scanning Pi. By the end of the project we had many different versions of the scanning script, proving we continuously improved the efficiency of the scans. We decreased how many commands we ran in our script in order to cut down on time, however, our scan does still run for a very long time. We started with a script that had about 8 runnable commands, and now our scanner runs one command automatically. This was a difficult decision for our group. We wanted our scanner to run relatively fast, but also to provide a lot of information. When we chose which scans to automate, we initially had two. As mentioned previously, "-p-" scans all 65535 possible ports. Because of this, scanning with "-p-" takes an extremely long time. We found that it was making our scan run almost 20 times slower and was giving us mixed results. The text files we would receive from the scan would have different output in them. When we first ran the scanner with "-p" included on the LCDI network, some of the text files were empty, some had errors in them, and two successfully scanned the network. To us, this did not make sense because it was the same script being run multiple times, but had different outputs. By taking out this command, our script now has a better success to failure ratio and also runs much faster, making it more efficient.

Further Work

We wish that we had more time in order to make our scanner have more functions, by added more working commands. Originally, we started with a script that required user input, but was capable of 8

commands. When we started to automate it, we realized that it was not possible for our scanner to keep all 8 of those commands. A lot of this was because some of the commands needed to be run under sudo (needed admin access), and we weren't sure how to make a sudo scan automated.

Appendix

Scan Times

Version No. (SHA-1 SUM)	Test No.	Days/Time (Hours:Mins:Seconds)	Comments
54a16e94bc	1	3/2/2018 / 00H:00M:37S	This was the first version of our script with 5 NMAP commands.
	2	3/2/2018 / 00H:00M:34S	(see first test comment)
	3	3/2/2018 / 00H:00M:35S	(see first test comment)
e40795282c	1	3/23/2018 / 02H:32M:32S	This was run for the time stated but had to be terminated early. There were no results gained from this scan of the Raspberry Pis.
	2	3/28/2018 / 03H:24M:22S	This was run for the time stated but lost its connection and because of that the scan was terminated. There were no results gained from this scan of the Raspberry Pi's.
	3	3/30/2018 / 00H:00M:36S	The host had to be manually set for this run and because of that it was a successful scan of the Raspberry Pi's.
8400244448	1	3/30/2018 / 00H:00M:36S	This scan was run successfully on the Raspberry Pi's
b165f4f56d	1	4/11/2018 / 00H:02M:42S	This script was run a total of 18 times on the LCDI network and the average time of all of the scans is given to the left. The longest scan time was 3 minutes 11 seconds while the shortest was 2 minutes 29 seconds.
	2	4/15/2018 / 02H:40M:56S	This run was a successful scan of the LCDI network.
	3	4/17/2018 / 41H:51M:23S	This run was a successful scan of the LCDI network.
	4	4/18/2018 / 02H:17M:42S	This script was run a total of 42 times and the average of all is the given time. The longest scan took:03:10:05 and the shortest took: 01:26:22

Script - main.py

```
1. #Network Scanning Basic Script
2. #!/usr/bin/python3.6
3. #For importing modules
4. import netifaces, netaddr, sys
5. #Import the other files from the Prints folder
6. from Prints import *
7.
8. from libnmap.process import NmapProcess
9. from libnmap.parser import NmapParser, NmapParserException
10. #starting nmap scan function
11.
12. def do_scan(target, options):
13.     #setting empty variable
14.     parsed = None
15.     #instantiating scan
16.     nmproc = NmapProcess(target, options)
17.     #running scan and setting it to a variable
18.     rc = nmproc.run()
19.     if rc != 0:
20.         print("Nmap scan failed: {0}".format(nmproc.stderr))
21.
22.     try:
23.         parsed = NmapParser.parse(nmproc.stdout)
24.     except NmapParserException as e:
25.         print("Exception raised while parsing scan: {0}".format(e.msg))
26.     return parsed
```

```
27. #starting sudo nmap scan function
28. def do_sudo_scan(target, options):
29.     #setting empty variable
30.     parsed = None
31.     #instantiating scan
32.     nmproc = NmapProcess(target, options)
33.     #running scan and setting it to a variable
34.     rc = nmproc.sudo_run()
35.     if rc != 0:
36.         print("Nmap scan failed: {0}".format(nmproc.stderr))
37.
38.     try:
39.         parsed = NmapParser.parse(nmproc.stdout)
40.     except NmapParserException as e:
41.         print("Exception raised while parsing scan: {0}".format(e.msg))
42.     return parsed
43.
44. if __name__ == "__main__":
45.
46.     if len(sys.argv) == 2:
47.         Interface = sys.argv[1]
48.     else:
49.         Interface = netifaces.interfaces()[1]
50.     #getting the ip
51.     target = netifaces.ifaddresses(Interface)[netifaces.AF_INET][0]['addr']
52.     mask =
    str(netaddr.IPAddress(netifaces.ifaddresses(Interface)[netifaces.AF_INET][0]['netmask']).netmask_bits(
    ))
53.
```

```
54.  #asking for what scan they would like to perform, commented out to make it automatic
55.  #options = input("What kind of scan would you like to perform? Input the NMAP option command\n")
56.
57.  #Sets the options to test the script becoming fully automated
58.  options = "-sV"
59.
60.  #The line below is if we want to scan an entire network. Commented out for testing purposes
61.  target = target+"/"+mask
62.
63.  #Basic port scan [Completed]
64.  if options == "-sV":
65.      #performing scan
66.      report = do_scan(target, options)
67.      #MAKE SURE THIS IS EMBEDDED IN ALL COMMAND STATEMENTS WITH YOUR SPECIFIC
     PRINT FUNCTION!!!!
68.      if report:
69.          #using print function to parse scan
70.          sV.print_scan_sv(report)
71.      else:
72.          print("No results returned.")
73.  # This one is for the automation
74.  elif options == "-sV -p-":
75.      #performing scan
76.      report = do_scan(target, options)
77.      #MAKE SURE THIS IS EMBEDDED IN ALL COMMAND STATEMENTS WITH YOUR SPECIFIC
     PRINT FUNCTION!!!!
78.      if report:
79.          #using print function to parse scan
80.          sV.print_scan_sv(report)
```

```
81.         else:
82.             print("No results returned.")
83.
84.     #Adds the ability to scan using TCP SYN scan *default* [Completed]
85.     elif options == "-sS":
86.         report = do_sudo_scan(target, options)
87.         if report:
88.             sS.print_scan_ss(report)
89.         else:
90.             print("No results returned")
91.
92.     #Adds the ability to scan UDP ports [Completed]
93.     elif options == "-sU":
94.         options = "-sU -p 123, 161, 162"
95.         report = do_sudo_scan(target, options)
96.         if report:
97.             sU.print_scan_su(report)
98.         else:
99.             print("No results returned")
100.         #Adds the ability to have a text file with hosts in it [Completed]
101.         elif options == "-iL":
102.             returned = iL.setup()
103.             options = returned[0]
104.             tar_list = returned[1]
105.             report = iL.do_scan(options, tar_list)
106.             if report:
107.                 iL.print_iL(report)
108.             else:
109.                 print("No results returned")
```

```
110.        #Add the ability to scan all the ports on a host [Completed]
111.        elif options == "-p-":
112.            report = do_scan(target, options)
113.            if report:
114.                aP.print_scan_p_(report)
115.            else:
116.                print("No results returned")
117.
118.    #Adds the ability to set version intensity [Completed]
119.        elif "--version-intensity" in options:
120.            report = do_scan(target, options)
121.            if report:
122.                sV.print_scan_sv(report)
123.            else:
124.                print("No results returned")
125.
126.        #Adds the ability to set version lightness [Completed]
127.        elif "--version-light" in options:
128.            report = do_scan(target, options)
129.            if report:
130.                sV.print_scan_sv(report)
131.            else:
132.                print("No results returned")
133.
134.        #Adds the ability to scan ip for protocols [Completed]
135.        elif options == "-sO":
136.            report = do_sudo_scan(target, options)
137.            if report:
138.                sO.print_scan_so(report)
```

```
139.                else:
140.                    print("No results returned")
141.
142.            #Adds the ability to scan 100 random hosts [Completed]
143.            elif options == "-iR 100":
144.                report = do_scan(target, options)
145.                if report:
146.                    iR.print_scan_ir(report)
147.                else:
148.                    print("No results returned")
149.
150.            #This will happen if none of the elif statements are triggered
151.            else:
152.                print("No valid options found")
153.                print(options)
```

Script - Prints/sV.py

```
1. def print_scan_sv(nmap_report):
2.     print("Starting Nmap {0} ( http://nmap.org ) at {1}".format(nmap_report.version,
   nmap_report.started))
3.     for host in nmap_report.hosts:
4.         if len(host.hostnames):
5.             tmp_host = host.hostnames.pop()
6.         else:
7.             tmp_host = host.address
8.
9.         if host.status == "up":
10.            print("Nmap scan report for {0} ({1})". format(tmp_host, host.address))
```

```
        print(" Port      STATE         SERVICE")

        for serv in host.services:
            pserv = "{0:>5s}/{1:3s}   {2:12s}  {3}". format(str(serv.port), serv.protocol, serv.state,
    serv.service)
            if len(serv.banner):
                pserv += " ({0})". format(serv.banner)
            print(pserv)
    print(nmap_report.summary)
```

References

Arr0way. "HighOn.Coffee." *Nmap Cheat Sheet*. N.p., 13 Dec. 2014. Web. 21 Mar. 2018.

Bister, Ronald. "Python-libnmap 0.7.0 : Python Package Index." *Python*. N.p., n.d. Web. 21 Mar. 2018.

Bister, Ronald. "Source Code for Libnmap.objects.cpe." *Libnmap*. N.p., n.d. Web. 23 Mar. 2018.

Bister, Ronald. "Source Code for Libnmap.process." *Libnmap*. N.p., n.d. Web. 21 Mar. 2018.

"Chapter 15. Nmap Reference Guide." *Nmap Network Scanning*. N.p., n.d. Web. 21 Mar. 2018.

Gite, Vivek. "Linux / UNIX: Change Crontab Email Settings (MAILTO)."*NixCraft*. N.p., 03 May 2017. Web. 23 Mar. 2018.

House, Nathan. "Nmap Cheat Sheet." *Station X*. N.p., 20 July 2017. Web. 21 Mar. 2018.

"Nmap Cheat Sheet." *HackerTarget.com*. N.p., 28 Aug. 2009. Web. 21 Mar. 2018.

"Python 3.6.5rc1 Documentation." *3.6.5rc1 Documentation*. N.p., n.d. Web. 21 Mar. 2018.

"Scheduling a Repetitive System Event (cron)."*Scheduling a Repetitive System Event (cron) (System Administration Guide, Volume 2)*. N.p., 2010. Web. 23 Mar. 2018.

Shrivastava, Tarunika. "29 Practical Examples of Nmap Commands for Linux System/Network Administrators." *Tecmint: Linux Howtos, Tutorials & Guides*. N.p., 3 Jan. 2015. Web. 21 Mar. 2018.

"User Guide - Installing Packages." *User Guide - Pip 9.0.2*. N.p., n.d. Web. 21 Mar. 2018.