Lab 2: Docker bash scripting

CNIT 47100

Group 11

Ethan Hammond

Date Submitted: 1/26/24

Date Due: 1/26/24

EXECUTIVE SUMMARY

In a medium sized organization, it is crucial to keep systems secure so that easy exploits from something like Kali Linux cannot disrupt your organization. As an offensive security specialist, it is necessary to learn effective bash scripting skills to be able to exploit organizations that do not efficiently secure their systems. Not securing systems will cost a company hundreds of thousands of dollars in repair and damage control. Offensive security professions should be able to write scripts to start docker instances, store docker container information, IP addresses, and format information correctly to be piped into scans. Nmap scans can be used to iterate through files and scan all IP addresses in the file. Scripts can be utilized to speed up this process and do a one-run port scan of all of an organizations docker containers. This is crucial to be aware of as a defensive or offensive security specialist.

STATEMENT OF WORK

The objectives of Lab 2 include the creation of bash scripts for lab testing of offensive security tactics. Project scope includes grabbing information about docker containers, active and inactive, parsing information between files and commands, and outputting commands to files to be analyzed by security analysts.

Objectives of Lab 2 include:

- Write a bash script to store docker container IDs
- Write a bash script to start all docker machines
- Create a bash cript to stop all docker machines
- Create a script to grab all docker IP addresses
- Pipe IP addresses into an Nmap scan
- Separate active docker containers from inactive containers
- Utilize Nmap to scan all TCP and UDP ports through a bash script.

The purpose of these created scripts is to train offensive security and automation without having to manually run scans on each individual docker container and store it in different files. Automation is incredibly important in offensive security as it allows for quick effective methods of exploiting systems.

PROCEDURES

The following procedures outline the steps taken to complete this project. For reference, *Italicized* words represent options, **bold** words represent buttons, and words in `Courier New` represent inputs that are typed.

Task 1 includes a proof of concept of a bash script automating pulling all docker images' container IDs and storing it in a file. Figures 1-8 show the created scripts to automate this process.

Task 1:

```
┌──(group11㉿g11kali)-[~]
└─$ sudo docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED      STATUS                     PORTS     NAMES
e80d3c653409   tleemcjr/metasploitable2:latest      "sh -c '/bin/service…"   5 days ago   Exited (134) 5 days ago              container-name
d8c98a9bc15c   santosomar/juice-shop                "docker-entrypoint.s…"   5 days ago   Exited (0) 5 days ago                juice-shop
d18e3f82de88   santosomar/rtv-safemode              "/run.sh"                5 days ago   Exited (137) 5 days ago              rtv-safemode
196a28f3d490   santosomar/mutillidae_2              "/run.sh"                5 days ago   Exited (137) 5 days ago              mutillidae_2
19a975263f1b   santosomar/dvwa                      "/main.sh"               5 days ago   Exited (137) 5 days ago              dvwa
c0332f0455de   santosomar/webgoat                   "/bin/sh -c '/bin/ba…"   5 days ago   Exited (137) 5 days ago              webgoat
e8d4361cd935   santosomar/dc31_02:latest            "/opt/druid/bin/star…"   5 days ago   Exited (1) 5 days ago                dc31_02
8d2f900d7091   santosomar/dc31_01:latest            "redis-server /etc/r…"   5 days ago   Exited (0) 5 days ago                dc31_01
32d3cbe01086   santosomar/yascon-hackme             "/start.sh"              5 days ago   Exited (137) 5 days ago              yascon-hackme
1179487d0b6d   santosomar/secretcorp-branch1        "/start.sh"              5 days ago   Exited (137) 5 days ago              secretcorp-branch1
8f05c62739e2   santosomar/dc30_02:latest            "/opt/solr/bin/solr …"   5 days ago   Exited (143) 5 days ago              dc30_02
1a6e45715bbf   santosomar/hackme-rtov               "nginx -g 'daemon of…"   5 days ago   Exited (0) 5 days ago                hackme-rtov
141247429e2f   santosomar/mayhem                    "nginx -g 'daemon of…"   5 days ago   Exited (0) 5 days ago                mayhem
6d93f275b733   santosomar/dvna                      "npm start"              5 days ago   Exited (0) 5 days ago                dvna
a7ef559509b9   santosomar/gravemind                 "/bin/sh -c /root/st…"   5 days ago   Exited (137) 5 days ago              gravemind
895c791340df   santosomar/dc31_03:latest            "/mnt/openfire/bin/o…"   5 days ago   Exited (143) 5 days ago              dc31_03
55be0997f43f   santosomar/hackazon                  "/bin/bash /start.sh"    5 days ago   Exited (137) 5 days ago              hackazon
dbf00fceb161   santosomar/ywing:latest              "/run.sh"                5 days ago   Exited (0) 5 days ago                Y-wing
80f6a7c92a1c   santosomar/galactic-archives         "./start.sh"             5 days ago   Exited (137) 5 days ago              galactic-archives
0472cb9fd2c0   santosomar/dc30_01:latest            "/usr/bin/entrypoint…"   5 days ago   Exited (0) 5 days ago                dc30_01
```

Figure 1: Current running docker images

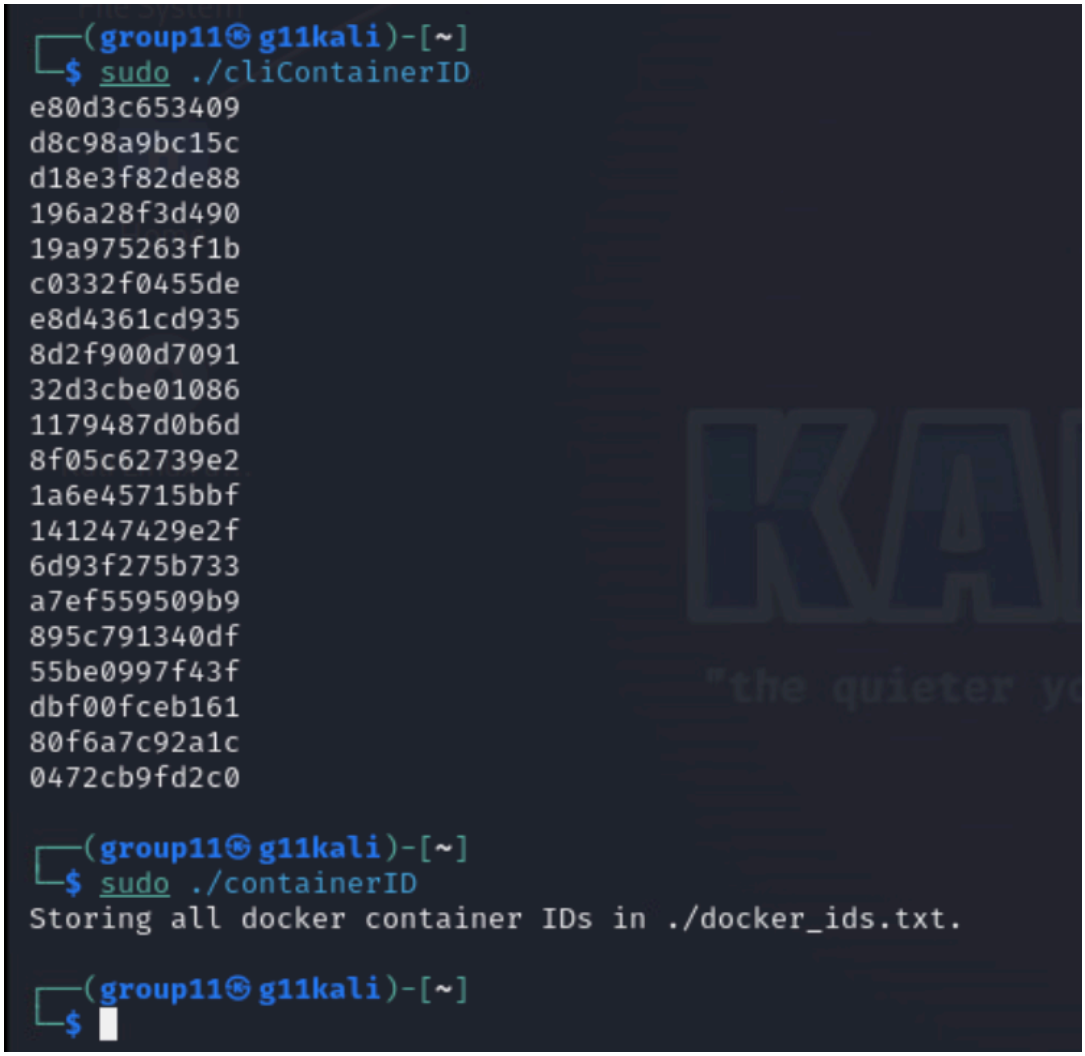

Figure 2: Bash script showing all IDs and bash script printing it to the screen

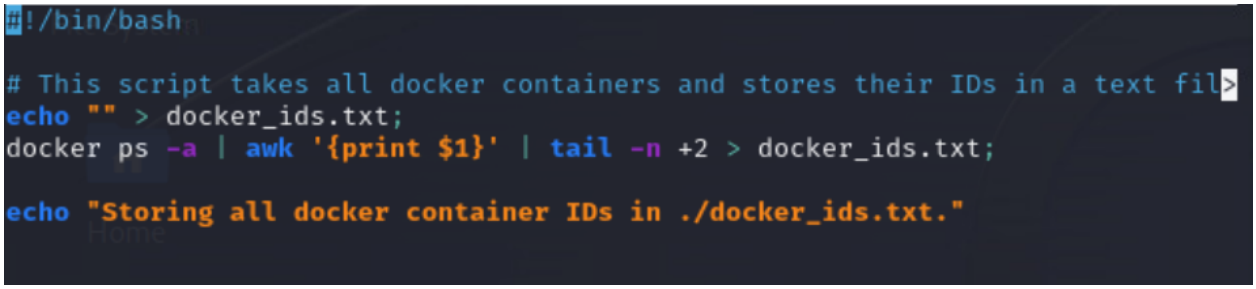

Figure 3: Bash script to put docker IDs in a file

```
#!/bin/bash

# This script takes all docker containers and stores their IDs in a text fil
echo "" > docker_ids.txt;
a="";
docker ps -a | awk '{print $1}' | tail -n +2;
```

Figure 5: Output of cliContainerID.sh

```
┌──(group11㉿g11kali)-[~]
└─$ sudo ./containerID
Storing all docker container IDs in ./docker_ids.txt
List of IDs:

e80d3c653409
d8c98a9bc15c
d18e3f82de88
196a28f3d490
19a975263f1b
c0332f0455de
e8d4361cd935
8d2f900d7091
32d3cbe01086
1179487d0b6d
8f05c62739e2
1a6e45715bbf
141247429e2f
6d93f275b733
a7ef559509b9
895c791340df
55be0997f43f
dbf00fceb161
80f6a7c92a1c
0472cb9fd2c0

┌──(group11㉿g11kali)-[~]
└─$
```

Figure 6: Concatenated scripts to accomplish

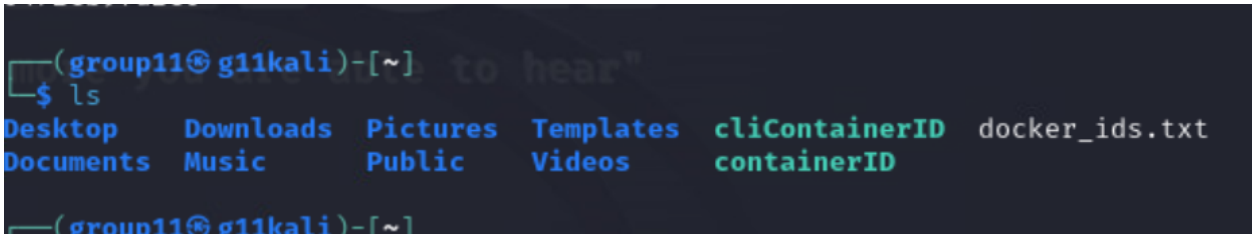

Figure 7: Docker_ids.txt file in home directory

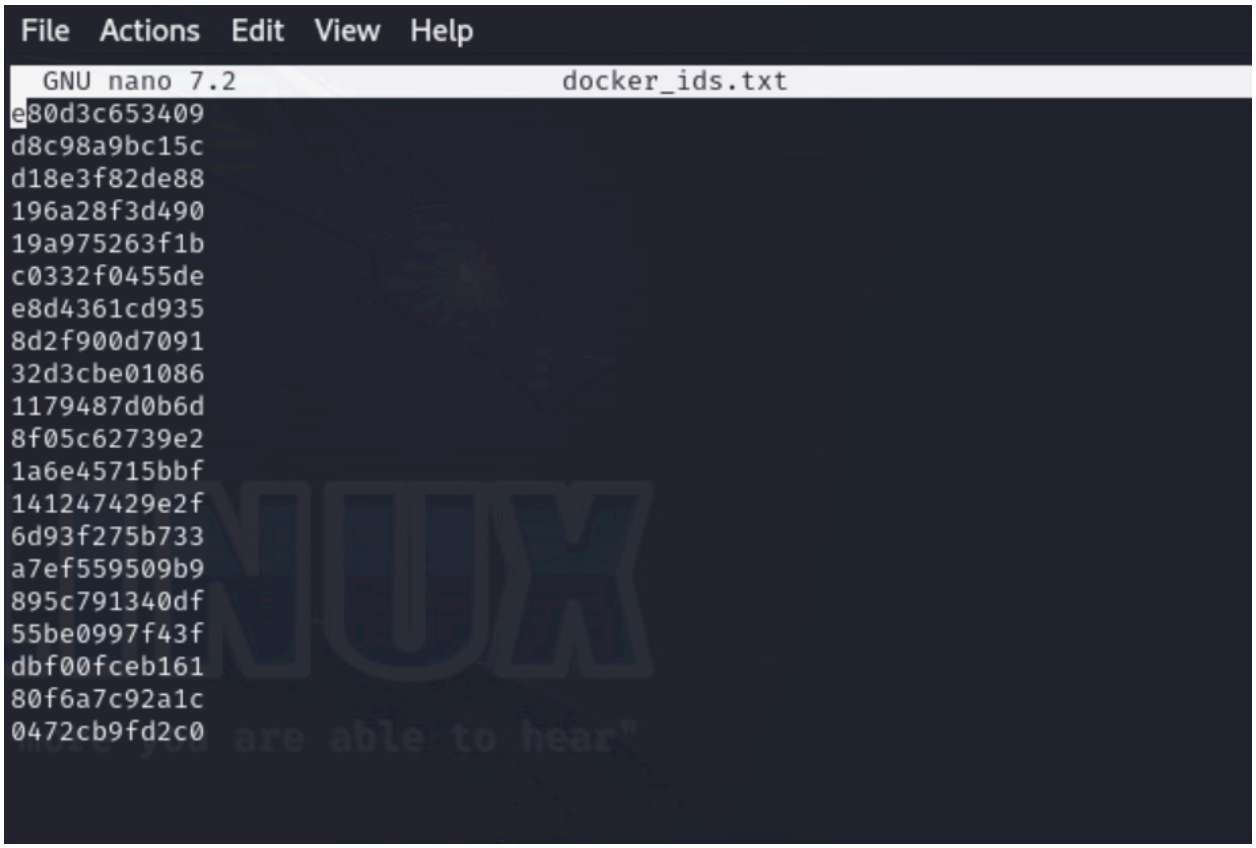

Figure 8: docker_ids.txt contents

Task 2:

Task 2 provides insight of bash starting all docker containers on the machine automatically based on container IDs from earlier scripts. Figures 9-10 show the docker container starting script.

```
  GNU nano 7.2                          dockerStart.sh
#!/bin/bash

# This script starts all docker machines by parsing containerID.sh

/home/group11/containerID.sh;
containerIDFile=/home/group11/docker_ids.txt;
echo & echo;
echo "Docker containers listed above are automatically starting now.";
echo & echo;
#docker start containerID
while IFS= read -r containerID; do
{
        echo "Container starting: $containerID";
        docker start "$containerID" > /dev/null 2>&1;
        sleep 2;
}
done < "$containerIDFile";
echo;
echo "All docker containers available are now running";
echo "Execution time: $SECONDS seconds";
```

Figure 9: dockerStart.sh script to start all docker containers in the text file.

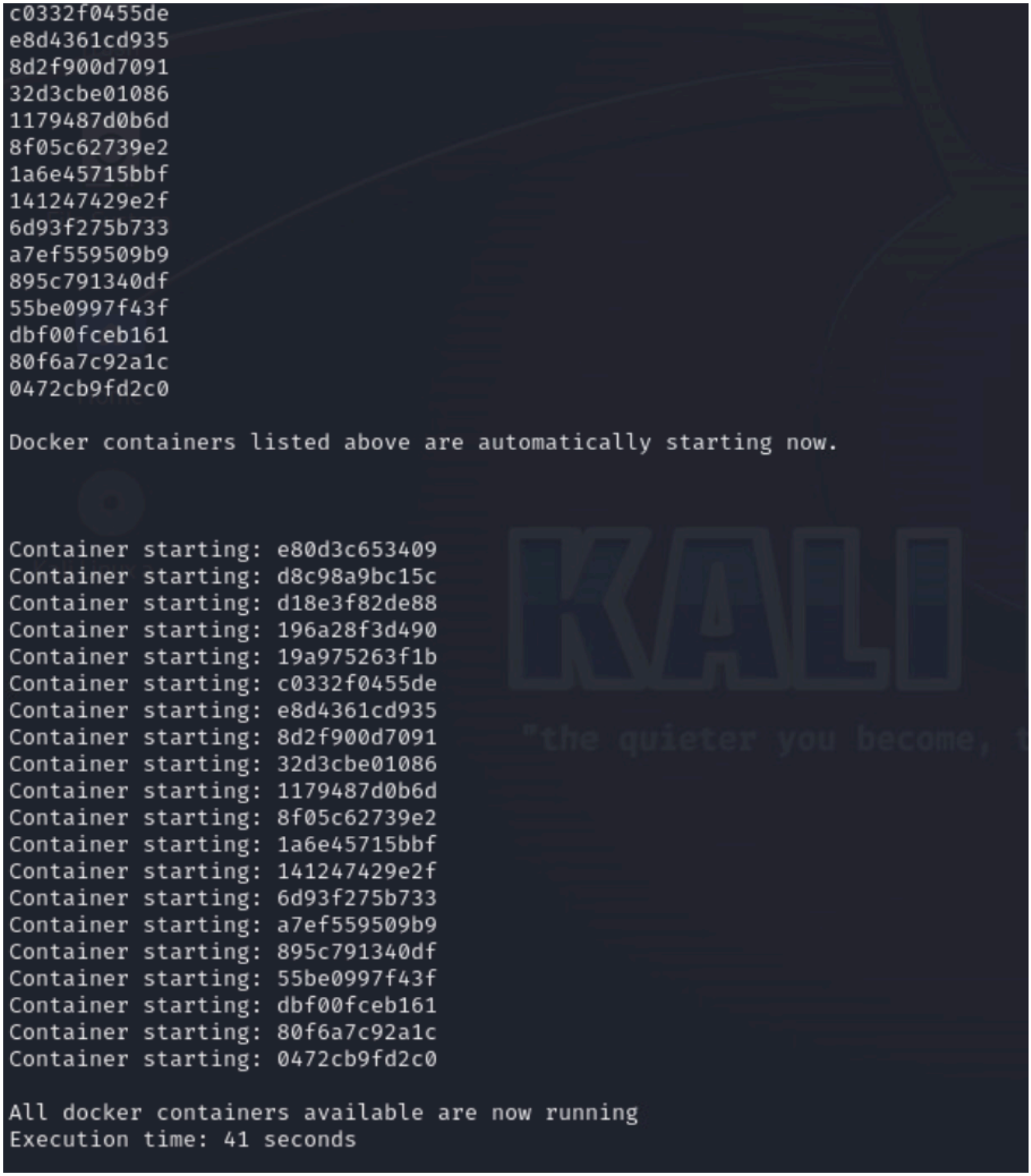

Figure 10: Output of dockerStart.sh

Task 3:

Task 3 outlines a similar objective of task 2 but stopping the docker containers instead of starting them. Figures 11-12 show the script content and output.

```
  GNU nano 7.2                          dockerStop.sh
#!/bin/bash

# This script stops all docker machines by parsing containerID.sh

/home/group11/containerID.sh;
containerIDFile=/home/group11/docker_ids.txt;
echo & echo;
echo "Docker containers listed above are being immediately stopped now.";
echo & echo;
#docker stop containerID
while IFS= read -r containerID; do      # While iterating file lines
{
        echo "Container stop: $containerID";
        docker stop "$containerID" > /dev/null 2>&1;    # Stop container
        sleep 2;        # Pause
}
done < "$containerIDFile";      # Finish loop
echo;
echo "All docker containers available are now stopped";
echo "Execution time: $SECONDS seconds";
```

Figure 11: dockerStop.sh script content

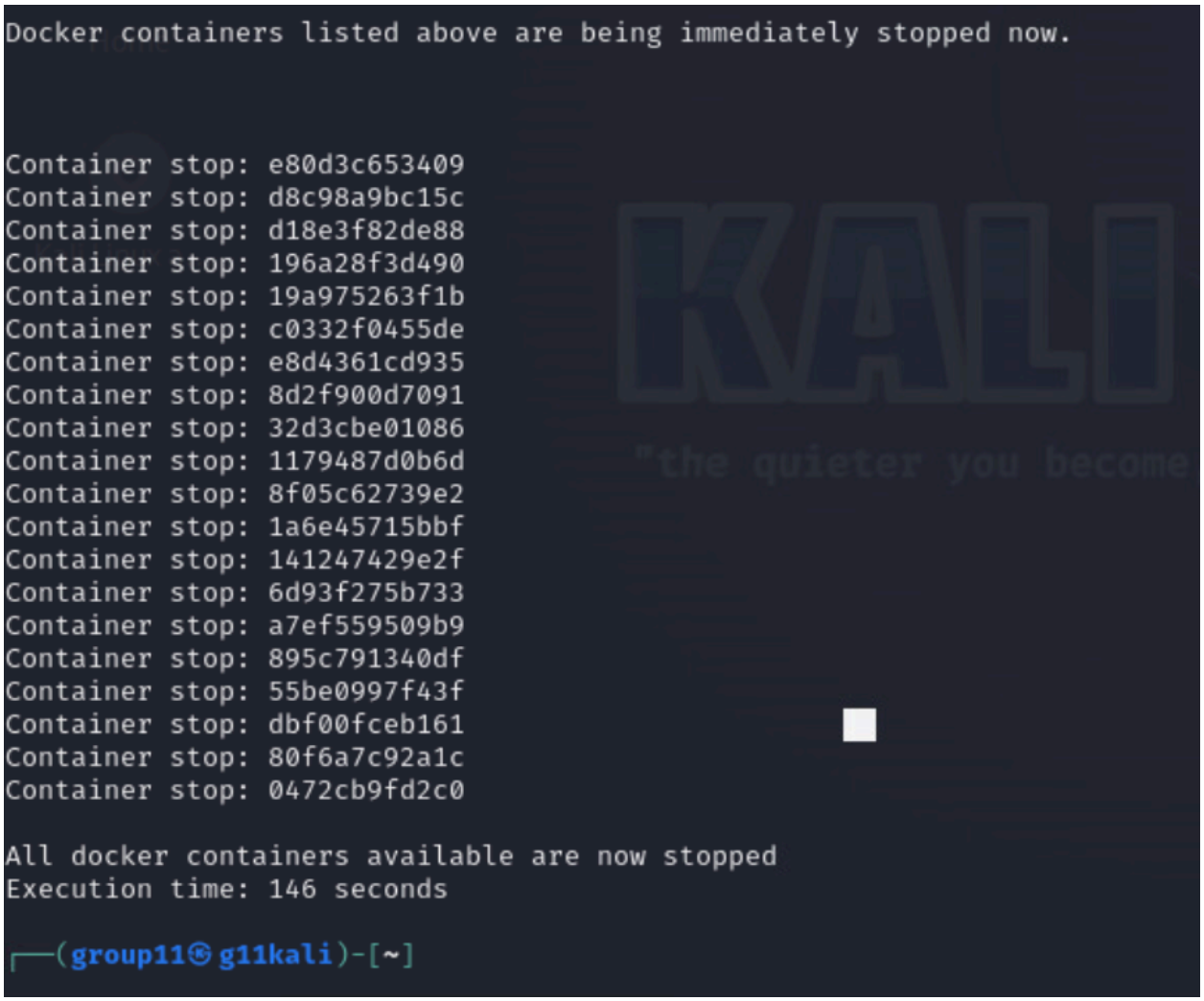

Figure 12: dockerStop.sh output

Task 4:

Task 4 creates the automation of grabbing IP addresses from the docker container IDs using the docker inspect command and storing the IP addresses in a host file. Figures 13-15 show the script contents and its output. The script grabs all IPs from the dockers trimming off gateways and external text leaving just the addresses behind.

Figure 13: containerIP.sh script

```
 GNU nano 7.2                                                containerIP.sh
#!/bin/bash

# This script takes all docker containers and stores their IDs in a text file called docker_ids.txt.
sudo rm /home/group11/docker_ips.txt;
touch /home/group11/docker_ips.txt;
sudo rm /home/group11/temp.txt;
touch /home/group11/temp.txt;
/home/group11/containerID.sh;

containerIPFile=/home/group11/docker_ips.txt;
containerIDFile=/home/group11/docker_ids.txt;
while IFS= read -r line
do      # While iterating file lines
{
        sudo docker inspect $line | grep -E -o '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' | sort | uniq >> /home/group11/temp.txt;
} done < "$containerIDFile";    # Finish loop

while IFS= read -r line2
do
{

grep -E -o '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' | grep -v -E '\b[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.1\b' | sort >> $containerIPFile

} done < /home/group11/temp.txt;
echo "Storing all docker container IPs in ./docker_ips.txt";
echo "List of IPs: ";
cat $containerIPFile;
echo;
```

Figure 13: containerIP.sh script

```
Storing all docker container IPs in ./docker_ips.txt
List of IPs:
10.6.6.11
10.6.6.12
10.6.6.13
10.6.6.14
10.6.6.15
10.6.6.16
10.6.6.17
10.6.6.18
10.6.6.19
10.6.6.20
10.6.6.21
10.6.6.22
10.6.6.23
10.6.6.24
10.6.6.25
10.6.6.26
10.7.7.21
10.7.7.22
10.7.7.23
172.17.0.2


┌──(group11㉿g11kali)-[~]
└─$
```

Figure 14: containerIP.sh execution output

Figure 15: Docker_ips.txt contents

Task 5:

Task 5 provides a proof of concept of scripts to scan docker images automatically via Nmap. This was done through the -iL flag in Nmap to make it iterate through a text file scanning all IP addresses. The script and its output into different file types can be seen in Figures 16-19 below.

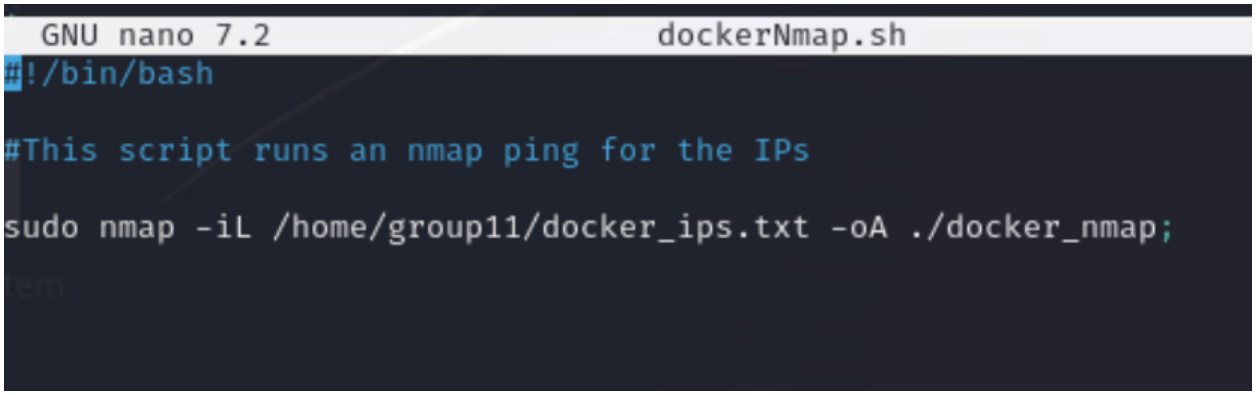

Figure 16: dockerNmap.sh script

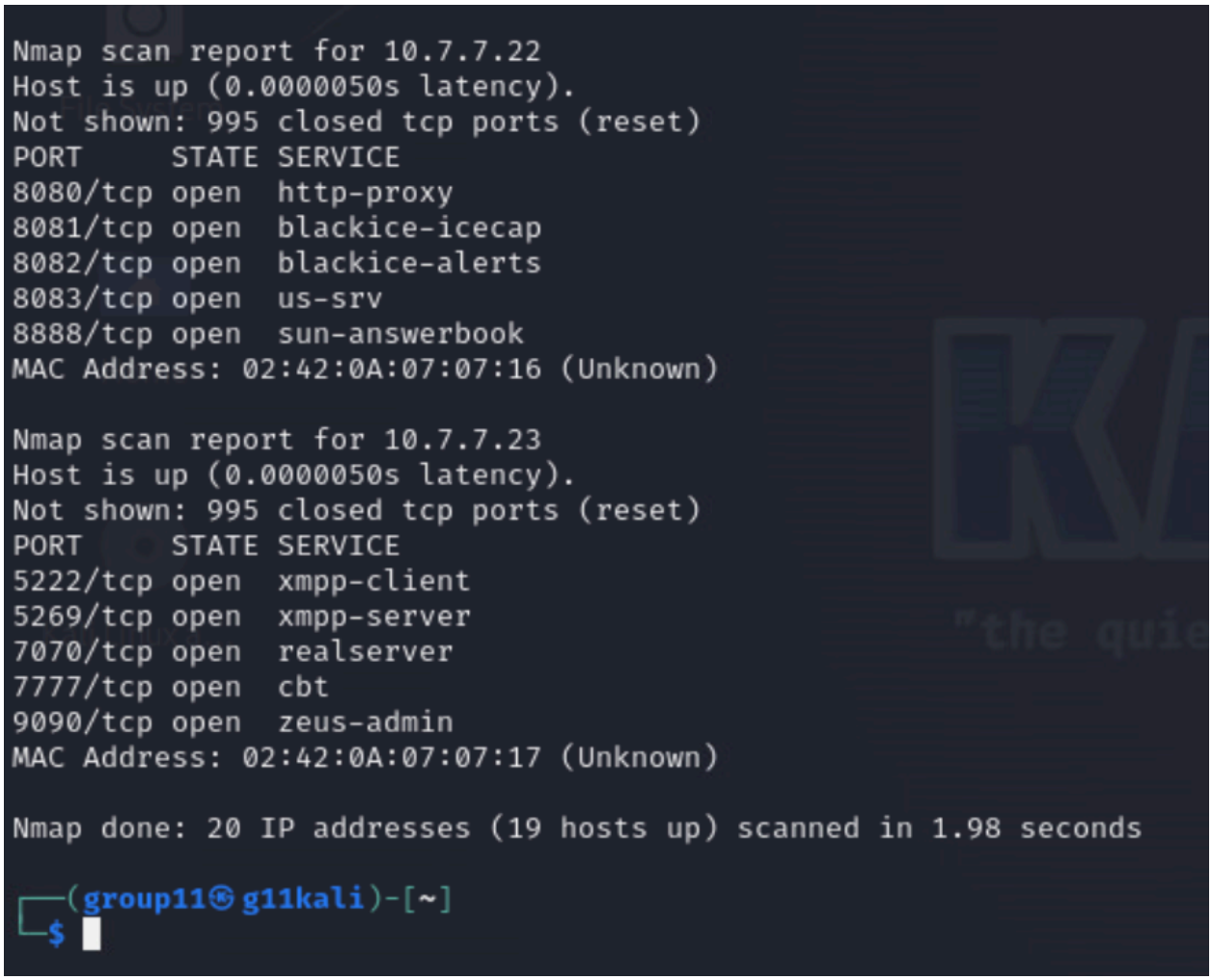

Figure 17: dockerNmap.sh output

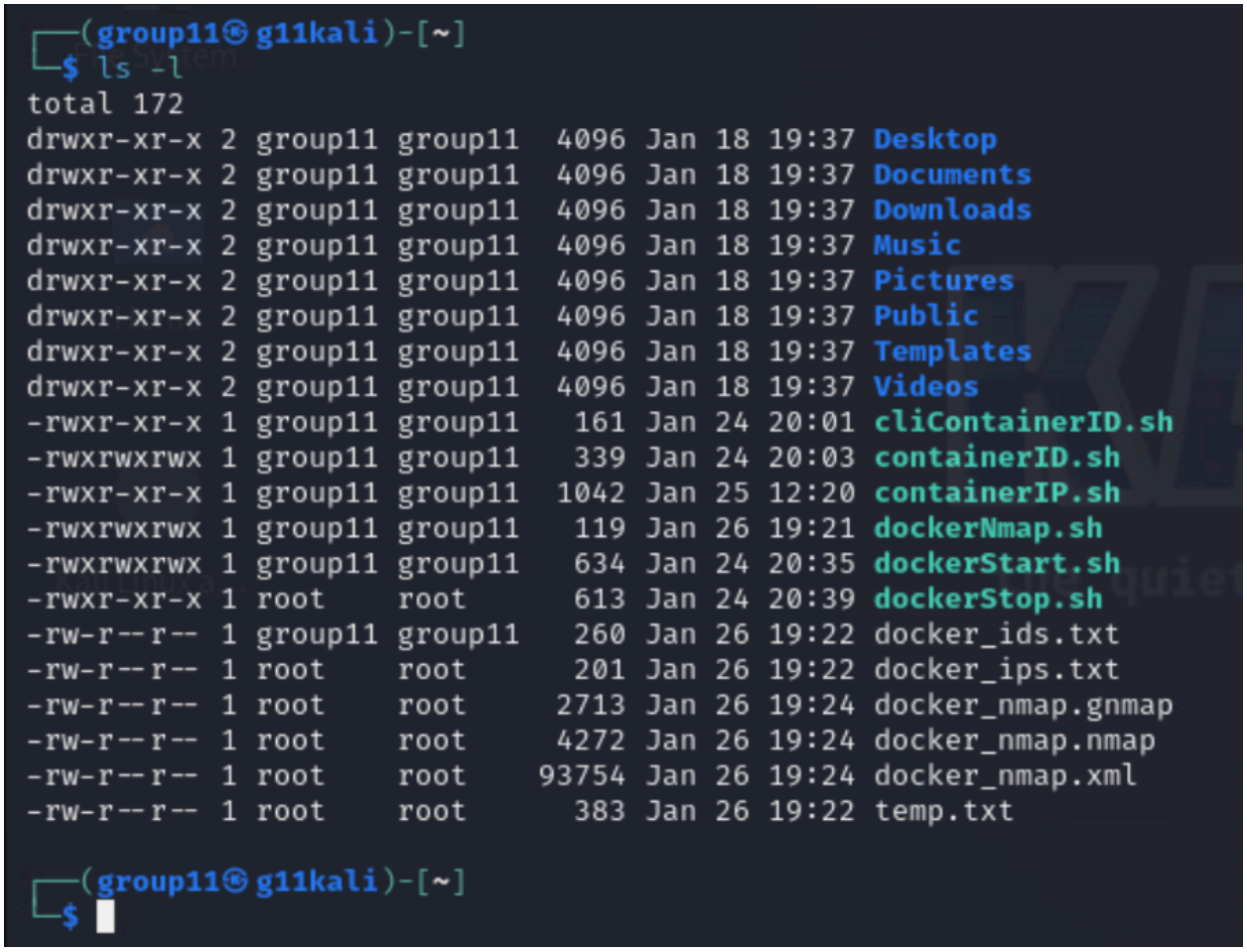

Figure 18: dockerNmap.sh files created in xml, nmap, and gnmap formats

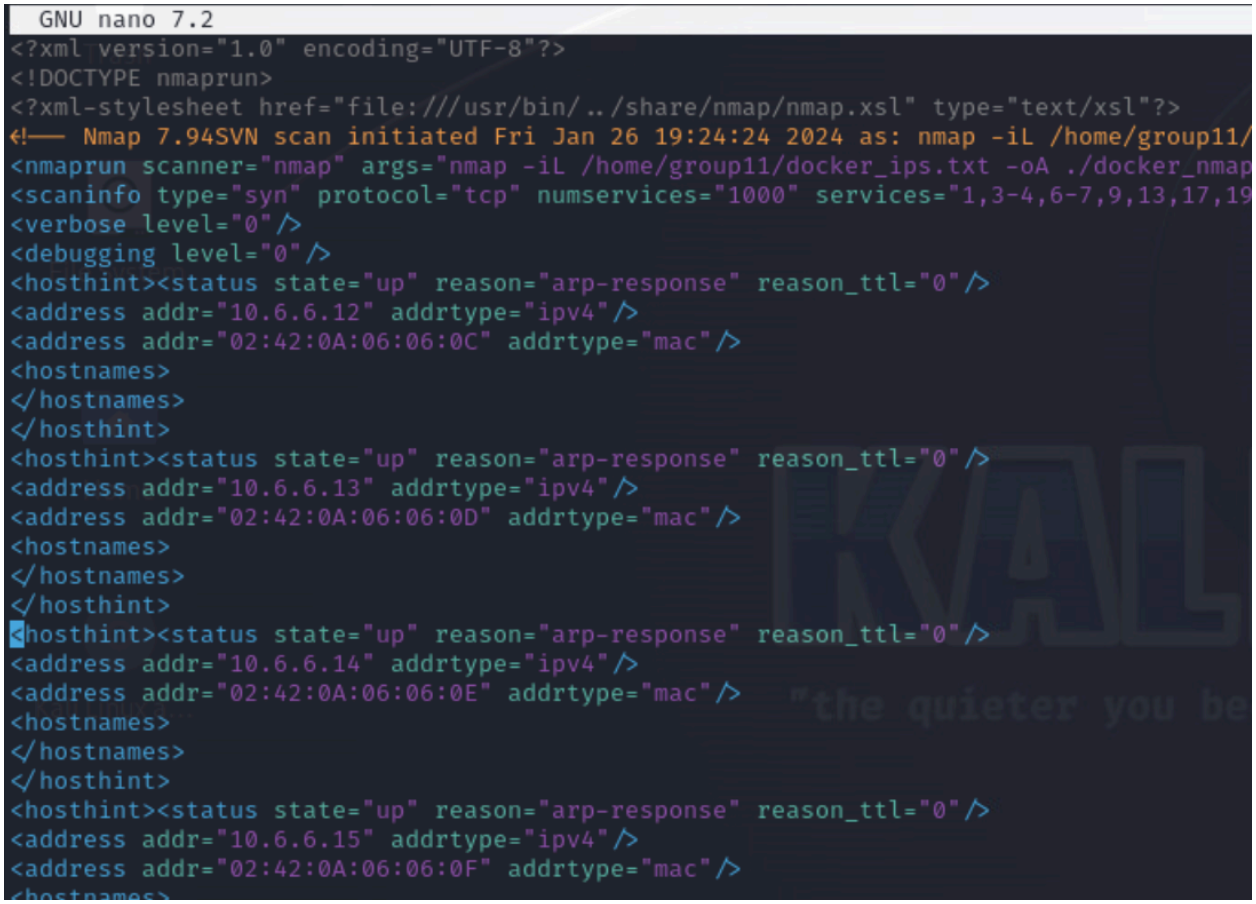

Figure 19: dockerNmap.xml output

Task 6:

Task 6 shows the process of bash scripts being able to separate active docker containers from inactive docker containers to just hold the live targets that can be attacked or exploited in the future. Figures 20-22 below outline the process and output of this script.

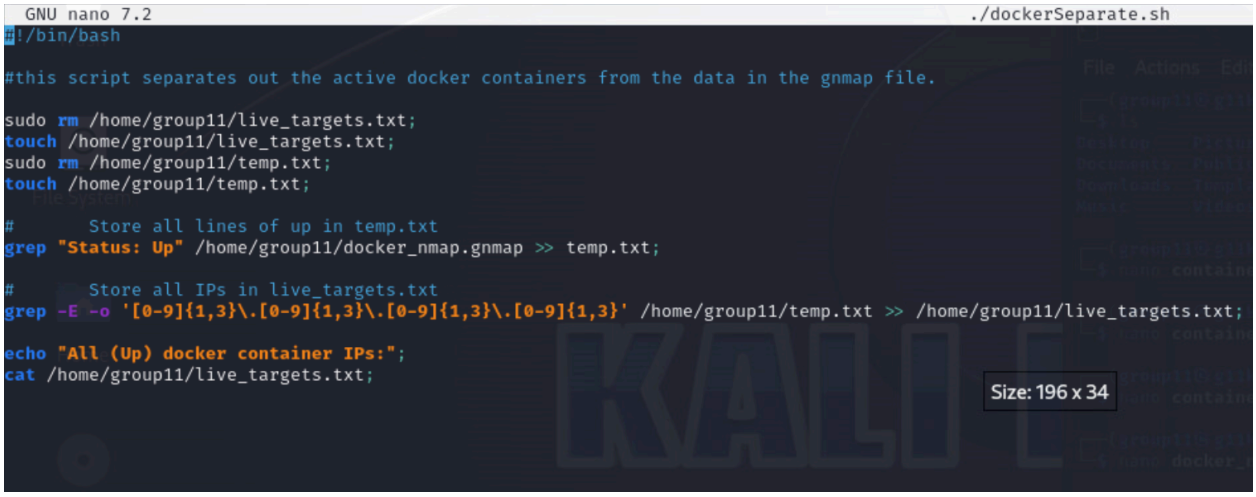

Figure 20: dockerSeparate.sh contents

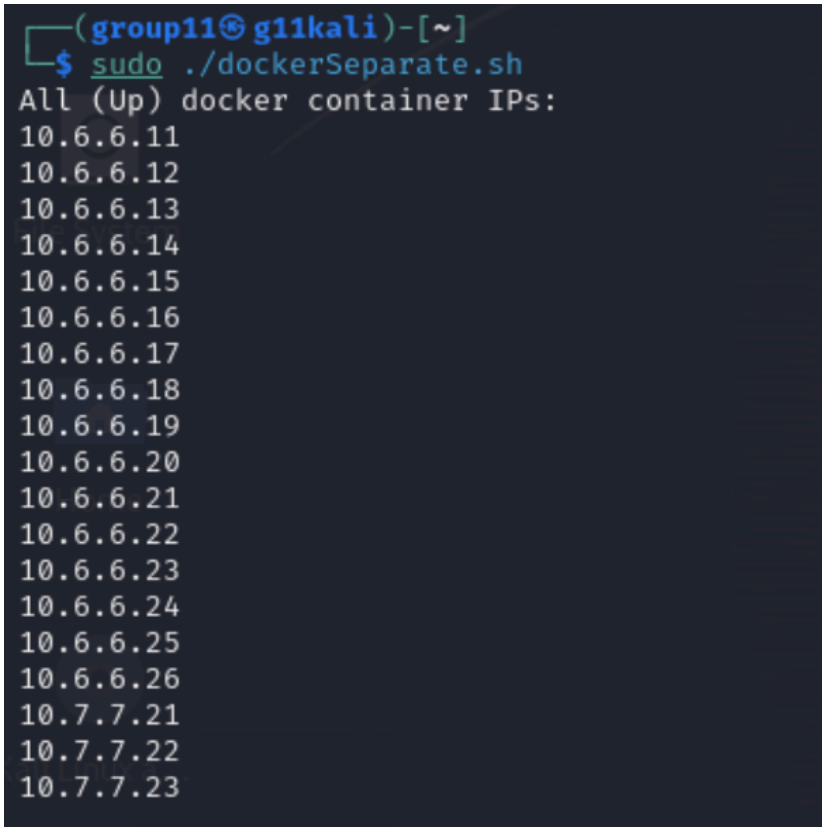

Figure 21: dockerSeparate.sh output into command line

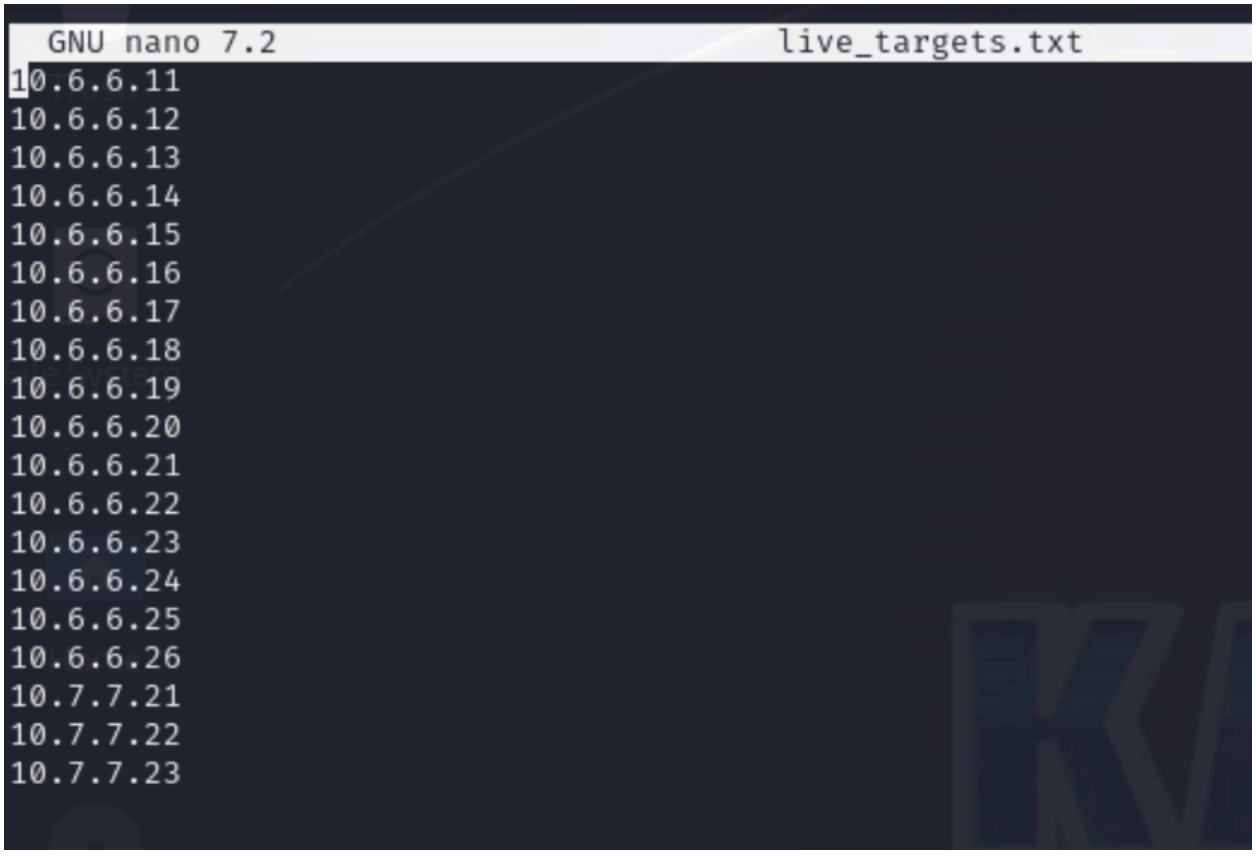

Figure 22: Live_targets.txt contents after running

Task 7:

Task 7 shows the process of using bash scripts to scan all TCP and UDP ports in all live target IP addresses through Nmap. The -sS, -sU, and -iL flags are used to iterate through the text file and scan all TCP and UDP ports in Figures 23-25 below.

```
 GNU nano 7.2                                  ./portScan.sh
#!/bin/bash

#This script uses nmap to port scan all IPs in the live_targets.txt file

sudo rm /home.group11/ports.txt;
touch /home/group11/ports.txt;

#       Scan all TCP and UDP ports and save in ports.txt
nmap -sS -sU -iL /home/group11/live_targets.txt >> /home/group11/ports.txt;
echo "Saving all UDP and TCP port scan output in ports.txt"
```

Figure 23: Contents of portScan.sh bash script

```
Nmap scan report for 10.7.7.22
Host is up (0.000026s latency).
Not shown: 1000 closed udp ports (port-unreach), 995 closed tcp ports (reset)
PORT     STATE SERVICE
8080/tcp open  http-proxy
8081/tcp open  blackice-icecap
8082/tcp open  blackice-alerts
8083/tcp open  us-srv
8888/tcp open  sun-answerbook
MAC Address: 02:42:0A:07:07:16 (Unknown)

Nmap scan report for 10.7.7.23
Host is up (0.000060s latency).
Not shown: 1000 closed udp ports (port-unreach), 995 closed tcp ports (reset)
PORT     STATE SERVICE
5222/tcp open  xmpp-client
5269/tcp open  xmpp-server
7070/tcp open  realserver
7777/tcp open  cbt
9090/tcp open  zeus-admin
MAC Address: 02:42:0A:07:07:17 (Unknown)

Nmap done: 19 IP addresses (19 hosts up) scanned in 2015.21 seconds
```

Figure 24: portScan.sh bash script output in ports.txt

Figure 25: Final directory content after all 7 scripts

CONCLUSIONS AND RECOMMENDATIONS

The takeaways from this lab are extensive knowledge increase in bash scripting, automation in Linux, and offensive security tactics. Even if a professional is going into something like defensive security, it is important for them to know the tactics of an offensive security professional. Using bash to automate, extract information, and exploit systems is a very important tactic to learn in an educational environment. Recommendations for this lab are to go above the requested tasks. Scripts can be extended in scope a small bit to make eligible for usage outside of the classroom environment. Adding features such as working REGEX parameters to be used on any IP address range instead of just the targeted docker container ranges can be very beneficial for a security specialist after the course is over and in the field. These scripts can not only be used in the class environment, but can also be used or tweaked for future use. Bash scripts for offensive security can be reused and carried on into a professional environment or for legal monetary benefit.

REFERENCES

Albano, W. by: J. (2023, June 28). *Silencing the output of a bash command*. Baeldung on Linux. https://www.baeldung.com/linux/silencing-bash-output

Docker Container start. Docker Documentation. (2024, January 5). https://docs.docker.com/engine/reference/commandline/container_start/

GeeksforGeeks. (2021, November 28). *Bash scripting - how to initialize a string*. GeeksforGeeks. https://www.geeksforgeeks.org/bash-scripting-how-to-initialize-a-string/

Matt ParkinsMatt Parkins 24.4k99 gold badges5050 silver badges6060 bronze badges. (1958, April 1). *GREP REGEX: List all lines except*. Stack Overflow. https://stackoverflow.com/questions/10189703/grep-regex-list-all-lines-except

Ramuglia, G. (2023, November 26). *GREP exclude: How to use -V to exclude words, patterns, or files in GREP*. Linux Dedicated Server Blog. https://ioflood.com/blog/grep-exclude-how-to-use-v-to-exclude-words-patterns-or-files-in-grep/

Shea, S. (2022, January 27). *How to use nmap to scan for Open Ports: TechTarget*. Security. https://www.techtarget.com/searchsecurity/feature/How-to-use-Nmap-to-scan-for-open-ports

Stork, V. (2021, September 13). *Bash sleep – how to make a shell script wait n seconds (example command)*. freeCodeCamp.org. https://www.freecodecamp.org/news/bash-sleep-how-to-make-a-shell-script-wait-n-seconds-example-command/

Vona, S. (2021, February 19). *How to get the execution time of shell scripts in linux*. Putorius. https://www.putorius.net/time-execution-of-script.html