Final Project Report

Name: Tyler Burleson Date: 4/22/2024

Summary

The purpose of this final project was to build a script to “automate log checking to identify attempted unauthorized logins to the server, identify the most egregious offenders, and add them to the servers firewall.” This script will help automate a tedious task of checking the auth.log file for multiple failed logins and manually adding each IP to the firewall to deny access. To accomplish this task, I had to utilize multiple technologies which I will list throughout this report.

To start breaking down this problem I needed to create an environment to test in. I personally utilized my Kali Linux virtual machine I already had created on my desktop for completing CTF’s. While I used this distribution of Linux, it did not rely on the architecture of Kali to accomplish. Instead, I could have used Ubuntu or another distro on an AWS instance, but for ease of use I decided to follow this route. Now that I had my environment for testing, I was ready to begin crafting this Bash script.

In the process of writing this script we were given a starting point which contained comments suggesting what functionality we should add. These included: error checking, common variables, colors, regex variables, and function goals. The requirements for this script included: gracefully catching any errors, ensuring the user is using sudo, display all unique IP addresses, display the detailed information of each IP address, add each offending IP address to the firewall utilizing UFW, display the current firewall rules, and finally offer a menu to allow the user to use each feature at their own leisure. Once I knew each of the requirements and had the starting pseudo code, the process of building the script was fairly simple.

The first step to building this script was ensuring I had UFW installed into my Kali Linux machine. This was just a simple command, “sudo apt-get install ufw.” Once this was installed, I ensured my firewall

was enabled and ran a few simple UFW commands. Once UFW was confirmed to be established correctly I started off in the script by creating the menu. I accomplished this by following the hints and creating an infinite loop that echoed to the console each option the user had access to. The keep it simple for the user, I utilized the numbers one through five for the navigation. Once I had the while loop that displayed the options, I needed to create a system to read in the user's input. By incorporating a switch statement, I was able to isolate the user's selection when they typed a given number and map that to its designated function as I completed each of them.

I started by completing the easiest function to me which was showing the firewall rules. Since I had experience with UFW before I knew this was a simple command, "sudo ufw status numbered." I then pipped the output into "less" to make it easier for the user to read. I then decided to work on some of the earlier error checking. I started with including two if statements at the beginning of the file to check if the user had "sudo" privileges and the next one to ensure that the script had an attachment to run against. Now that I had the simpler functionality completed along with the color variables I decided to start tackling the more complicated functions. Since a majority of the functions coming up needed to be able to read IP's I started by building the regex for finding a qualifying IP. Since I had some of my old labs from my Unix class, I decided to break out my regex into an initial quad then combined them into the format on an IP address.

Now that I had my regex to identify a valid IP address, I was able to start reading in the file the user provided. I started with the function of displaying the unique IP addresses. I knew this function was going to be fairly straight forward and the only issues I ran into during this section was having to review using "awk." Besides awk, I knew I needed to use the grep command, sort the output, and use the unique command with -c to count. To accomplish this functions goal I implemented the commands previously stated pointing at "rhost=$IP" and "$fileToUse." By following the hint in the starter code, I was able to remove the duplicate entries in the "$fileToUse" and isolate the first record of the IP connected

to the “rhost” text. Once I had the unique IP’s I moved onto the function that identified offending IP’s with 100+ connection attempts. This function was almost identical to the previous one, but I knew I needed to include a conditional to check the number of attempts. I accomplished this by using the awk command and including “{if ($1 >= 100) print $0}.” I originally had the conditional check for “if $1 > 99,” but this broke the function, and I honestly don’t know why. I couldn’t find a common denominator that made since, but what I did notice was that if the IP contained “99” anywhere inside it, it registered it as an offender. I took this error and tried to troubleshoot it with the help of Blackbox, and ultimately came to the current code conclusion which worked.

Now that I had completed gathering the IP’s from the attached file I was able to start using that data to perform other functions. I took the offending IP’s from the earlier function and used it inside our new function to add them to the firewall using UFW. This was simple by itself, but I needed to add some error checking to ensure the IP wasn’t a duplicate inside the system. I did this by using a for loop with each item included in the offending IP’s and used a nested if to check if it was inside already. If it wasn’t, it would add it to the firewall. If it was already then it would say “Skipping this rule.” I originally ran into an issue where it would use the default error of adding an existing rule. This took some troubleshooting, but finally found out It was due to the way I was setting up my “$val” in each offending IP. Once I had the output formatted correctly, the conditional was able to successfully check the firewall and skip the existing rule.

The final feature I needed to add was simple when coded, but I was unsure how to gather the data. After some google searches and confirmation from my friends who had taken the course previously, they told me to use ipinfo.io to get greater detail on each entry. Once I had a token to utilize the API, I read the documentation on how to call the API through the terminal and found it was a simple curl command. Once I had this, I added the curl command into the function, echoed some additional

information, and pipped the output into less to make it easier to read. Once I had each of the functions mapped properly to the switch statement and tested the script I completed with final project.

Screen Shots

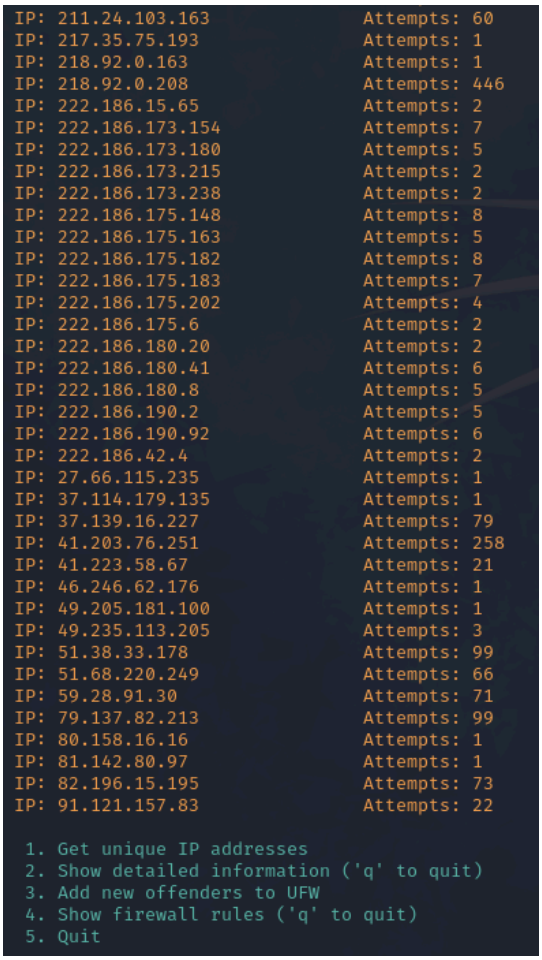


Figure 1 (List of unique IP's)

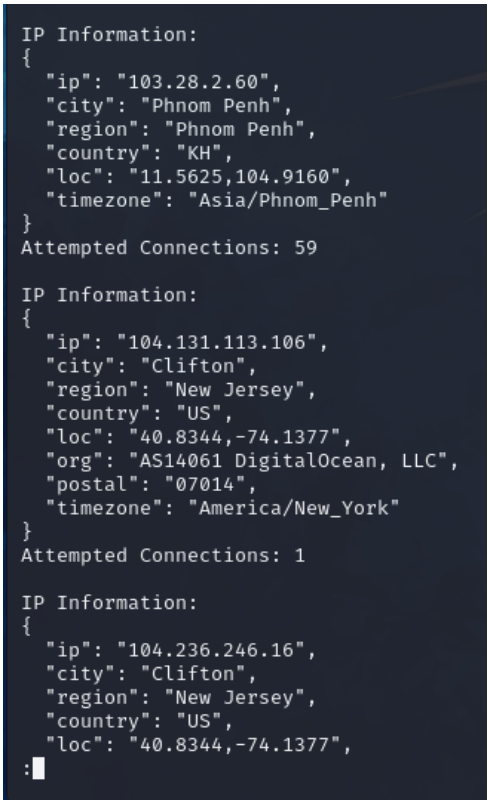


Figure 2 (High detail list of the IP's)

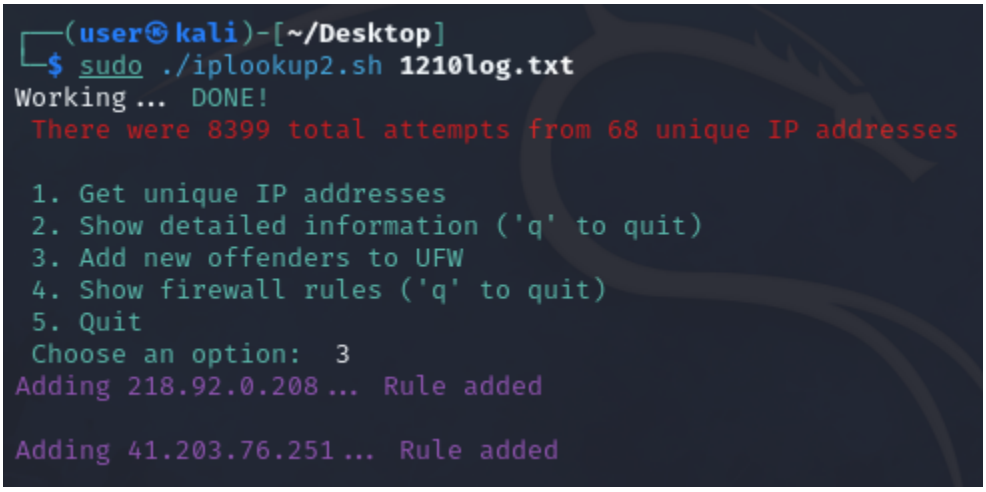


Figure 3 (Adding new offending IP's to the firewall if they don't exist)

```
 1. Get unique IP addresses
 2. Show detailed information ('q' to quit)
 3. Add new offenders to UFW
 4. Show firewall rules ('q' to quit)
 5. Quit
 Choose an option:  3
Adding 218.92.0.208 ...  Skipping adding existing rule

Adding 41.203.76.251 ...  Skipping adding existing rule
```

Figure 4 (Adding a offending IP to the firewall if it does exist already)

```
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] Anywhere                   DENY IN     1.4.149.136
[ 2] Anywhere                   DENY IN     1.54.92.152
[ 3] Anywhere                   DENY IN     101.108.212.229
[ 4] Anywhere                   DENY IN     101.32.222.206
[ 5] Anywhere                   DENY IN     103.133.123.220
[ 6] Anywhere                   DENY IN     103.147.40.22
[ 7] Anywhere                   DENY IN     103.226.249.187
[ 8] Anywhere                   DENY IN     103.47.81.151
[ 9] Anywhere                   DENY IN     103.73.102.247
[10] Anywhere                   DENY IN     104.131.88.229
[11] Anywhere                   DENY IN     106.51.77.52
[12] Anywhere                   DENY IN     110.136.57.96
[13] Anywhere                   DENY IN     110.77.235.167
[14] Anywhere                   DENY IN     113.176.98.128
[15] Anywhere                   DENY IN     113.184.185.171
[16] Anywhere                   DENY IN     113.190.233.246
[17] Anywhere                   DENY IN     115.85.53.91
[18] Anywhere                   DENY IN     116.193.136.108
[19] Anywhere                   DENY IN     116.58.236.84
[20] Anywhere                   DENY IN     117.222.38.126
[21] Anywhere                   DENY IN     117.5.146.39
[22] Anywhere                   DENY IN     118.173.232.224
[23] Anywhere                   DENY IN     118.96.192.76
[24] Anywhere                   DENY IN     119.204.96.131
[25] Anywhere                   DENY IN     122.51.28.170
[26] Anywhere                   DENY IN     125.161.252.110
[27] Anywhere                   DENY IN     125.24.194.36
[28] Anywhere                   DENY IN     138.68.110.166
[29] Anywhere                   DENY IN     138.99.250.214
[30] Anywhere                   DENY IN     139.217.84.94
[31] Anywhere                   DENY IN     14.162.191.203
```

Figure 5 (List of firewall rules)

Research Questions

1. Run less -N 1210log.txt
   a. Look at line # 1271. What do you think is going on here? Does this look like an attempted hack? It looks like a cron job. While we can't confirm if this is an attempted hack, we can confirm what it is if we investigate deeper.

   b. Look at line # 9044. Is this a hacking attempt? What is the username? Yes "cronjob"

   c. Look at line # 824. What is the offending IP address, port, and username? 218.92.0.208 port-32145 user: root

2. Write out the structure for a BASH while loop. Use # code goes here to indicate where the executed code is placed in the loop

```
while [ # Code goes here for the condition to run ] ; do
        # Code goes here
done
```

3. Write out the structure for a BASH if/then/else block. Use # code goes here to indicate where the executed code is placed in the block

```
If [# Code goes here for the condition to run]; then
        # Code here
Else
        # Code here
fi
```

4. Write out the structure for a BASH case block including a default value, again indicating where the code would appear within the block's structure

```
Case $option in
        "# code here for the option")
                # Code here
                ;;
        "# code here for the next option")
                # Code here
                ;;
        "*") # this is the default case
                # Code here
                ;;
esac
```

5. Run wget https://csci2200.net/downloads/baddies.txt

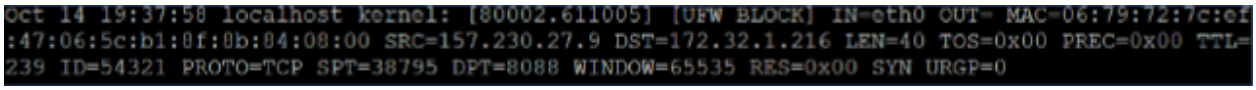
(This is a log file from another script that contains line number, # of attempts, and 'username.' It is a record of the user names that the bad guys have tried to use to crack the system and the number of times each username has been used. Run less baddies.txt. To search for a string with less, enter the front slash ('/') and the string

a. What is the username for line 1366? (/1366 ... 'g' returns to the top of the file. I don't know why it isn't wrapping, but you'll need to start the next search from the top of the file (see (b))
The username is 'ubuntu.'

b. Why would someone try to use the username at line # 1101 to hack a system?

The username for line 1101 is 'root.' This is a common tactic for attackers to target "root" since all Linux distributions have this super user. If the attacker can access this account, they usually have full administrator privileges inside the system and can cause havoc. Additionally, by targeting this account, the attacker doesn't have to guess other user accounts.

6. The following screen shot is a line of output from /var/log/syslog (another system log file)



a. What is happening here, i.e., what is this output telling you happened? – UFW is denying a TCP packet
b. What is the originating IP address? – 157.230.27.9
c. Which protocol was being used – TCP

7. Our iplookup.sh script is interactive, which I think is pretty cool. But what if we wanted to add it to cron and set it to run and populate the firewall automatically. How could we modify the script to accomplish this? (You don't actually have to modify the script. Just describe how you could go about it - "fire and forget"

To achieve this, we would start by removing the interactive menu and make the script execute immediately. Next, we would only want the script to analyze the IP's from the log file with 100+ attempts using the same grep statement. Once that part is done, then add those offenders to the firewall. Essentially, we remove all the quality-of-life stuff for the user to interact with and gather more information and leave it to its core grep and add commands.